รายงาน

เรื่อง

Program สำหรับการ Train Multilayer Perceptron โดยใช้ Genetic Algorithms

Wisconsin Diagnostic Breast Cancer (wdbc.data)

โดย

น.ส. ภัทรจาริน ผดุงกิจเจริญ รหัสนักศึกษา 650610851

เสนอ

รศ.ดร. ศันสนีย์ เอื้อพันธ์วิริยะกุล

รายงานนี้เป็นส่วนหนึ่งของรายวิชา CPE 261456

Introduction to Computational Intelligence

สาขาวิชาวิศวกรรมหุ่นยนต์และปัญญาประดิษฐ์

ภาคเรียนที่ 1 ปีการศึกษา 2567

มหาวิทยาลัยเชียงใหม่

1.ลักษณะการทำงานของระบบ

1.1 วัตถุประสงค์

ระบบนี้มีวัตถุประสงค์เพื่อสร้างและฝึกโมเดล Multilayer Perceptron (MLP) โดยใช้ Genetic Algorithm (GA) ให้ทำการทดลองกับ wdbc.data (Wisconsin Diagnostic Breast Cancer (WDBC) จาก UCI Machine learning Repository) โดยที่ data set นี้ มี 2 classes และ 30 features ซึ่งในแต่ละ sample จะมีทั้งหมด 32 ค่าโดยที่

1) ID number

2) Diagnosis (M = malignant, B = benign) ⟶ class

3-32) เป็นค่า features ทั้ง 30

ให้ทำการทดลองโดยใช้ 10% cross validation เพื่อทดสอบ validity ของ network ที่ได้ และให้ทำการเปลี่ยนแปลงจำนวน hidden layer และ nodes

1.2 ขั้นตอนการทำงาน

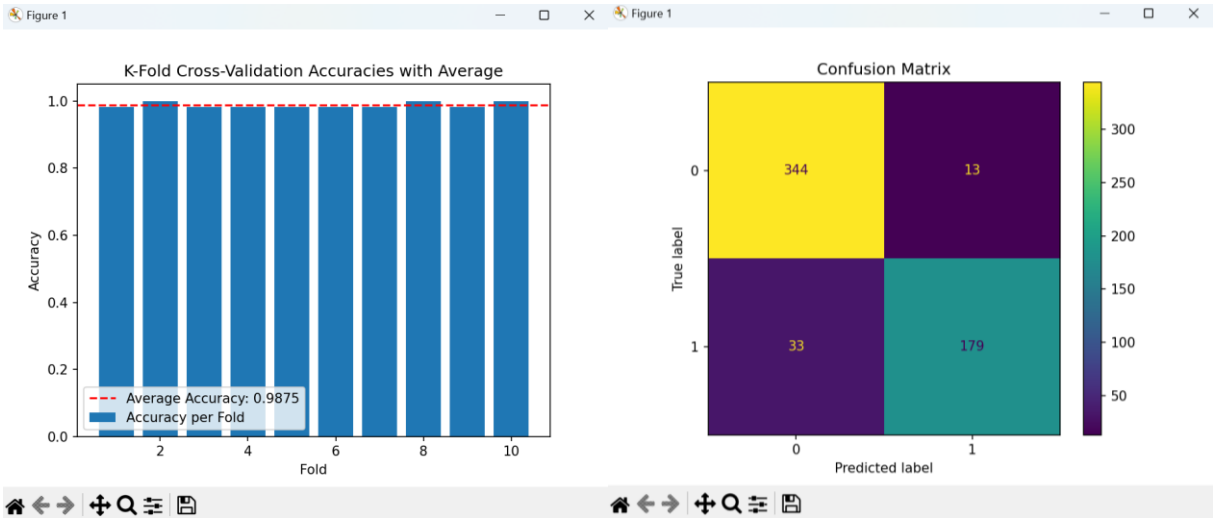
- **โหลดข้อมูล:** ระบบจะโหลดข้อมูลจากไฟล์ wdbc.data.txt ซึ่งประกอบด้วยฟีเจอร์และป้ายชื่อ (label) ที่ใช้ในการจำแนกประเภทเซลล์มะเร็ง
- **การเตรียมข้อมูล:** ฟีเจอร์จะถูกปกติฟีเจอร์ (normalize) และแบ่งข้อมูลออกเป็นชุดฝึก (training set) และชุดทดสอบ (validation set) โดยใช้วิธี k-fold cross-validation
- **สร้าง MLP:** ระบบจะสร้างโมเดล MLP ที่มีโครงสร้างสามารถปรับจำนวน hidden layers และ nodes ได้
- **การปรับแต่งโมเดลด้วย GA:** Genetic Algorithm จะถูกใช้ในการปรับแต่งน้ำหนักของโมเดล MLP โดยการสร้างประชากรเริ่มต้น ทำการคัดเลือก การข้ามพันธุ์ และการกลายพันธุ์เพื่อหาโมเดลที่ดีที่สุด
- **ทดสอบโมเดล:** โมเดลที่ดีที่สุดจะถูกทดสอบกับชุดทดสอบเพื่อวัดความแม่นยำ
- **แสดงผล:** ระบบจะทำการแสดงความแม่นยำของโมเดลและแสดง confusion matrix เพื่อวิเคราะห์ผลการจำแนกประเภท

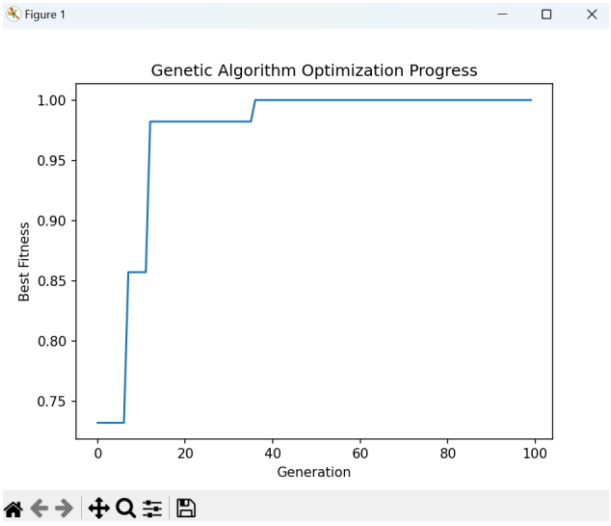
2. Simulation ของระบบ ผลการทดลอง และวิเคราะห์

2.1 การตั้งค่าการทดลอง และผลการทดลอง

ในการทดลองนี้ โครงสร้างของเครือข่ายประสาทแบบหลายชั้น (MLP) ถูกทดสอบกับโครงสร้างสองรูปแบบที่แตกต่างกันในจำนวนเลเยอร์และจำนวนโหนด (nodes) ในแต่ละเลเยอร์:

hidden_layers = [128] : โครงสร้างนี้ใช้เพียงเลเยอร์เดียวที่ประกอบด้วยโหนดจำนวน 128 โหนด

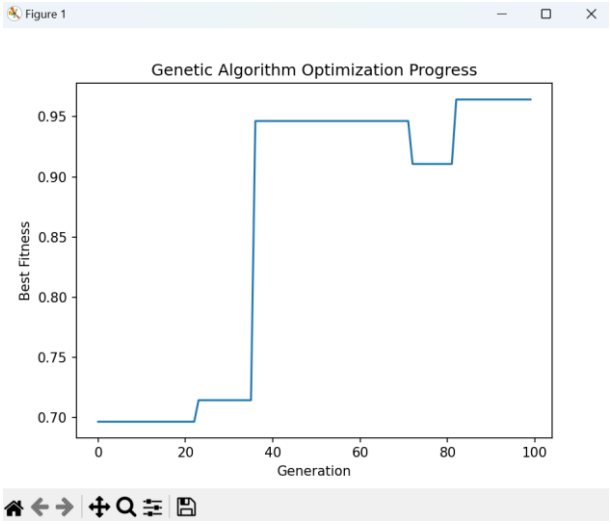




ผลการทดลอง: โมเดลแบบ single-layer ที่มีจำนวน nodes สูงมีความแม่นยำดีขึ้นในบางกรณี แต่มีแนวโน้มที่จะ overfit โดยเฉพาะเมื่อข้อมูลตัวอย่างมีน้อย

การวิเคราะห์: โมเดลนี้อาจเหมาะกับการจำแนกที่ฟีเจอร์มีความสัมพันธ์สูง แต่ขาดความยืดหยุ่นในกรณีที่ต้องการแยกแยะฟีเจอร์ที่ซับซ้อนมาก

hidden_layers = [64, 32, 16] : โครงสร้างนี้ใช้เลเยอร์หลายชั้นที่ประกอบด้วยโหนด 64, 32 และ 16 ตามลำดับ



ผลการทดลอง: โมเดลแบบ multi-layer แสดงความแม่นยำที่ดีกว่าในการจำแนกประเภทและมีความยืดหยุ่นมากขึ้น ส่งผลให้ลดความเสี่ยงต่อการ overfit

การวิเคราะห์: การลดจำนวน nodes ในแต่ละชั้นช่วยให้โมเดลสามารถประมวลฟีเจอร์ที่แตกต่างกันได้ดีขึ้น เพิ่มความสามารถในการปรับตัวและลด overfitting

2.2 การวิเคราะห์ผลการทดลอง

จากผลการทดลองข้างต้นสามารถวิเคราะห์ได้ดังนี้:

ผลกระทบของจำนวนเลเยอร์และโหนด: โมเดลที่มีหลายเลเยอร์ (hidden_layers = [64, 32, 16]) สามารถทำให้ความแม่นยำในการจำแนกประเภทสูงกว่าแบบเลเยอร์เดียว เนื่องจากโครงสร้างหลายชั้นช่วยให้โมเดลสามารถเรียนรู้ข้อมูลที่ซับซ้อนและจับรายละเอียดได้ดีกว่า อย่างไรก็ตาม การใช้เลเยอร์หลายชั้นมีข้อเสียคือทำให้การคำนวณซับซ้อนมากขึ้นและใช้เวลาในการฝึกนานกว่า

ประสิทธิภาพของ Genetic Algorithm (GA): การใช้อัลกอริทึม GA เพื่อปรับแต่งน้ำหนักของโมเดล MLP นั้นมีผลในเชิงบวกต่อความแม่นยำในการจำแนกประเภท โดยทำให้โมเดลสามารถหาค่าที่เหมาะสมสำหรับการจำแนกเซลล์มะเร็งได้อย่างมีประสิทธิภาพ การคัดเลือกและการผสมพันธุ์ช่วยในการสร้างรูปแบบที่หลากหลายและการกลายพันธุ์ช่วยในการป้องกันไม่ให้โมเดลติดอยู่ในค่าที่ไม่เหมาะสม

ข้อดีของการใช้ cross-validation: การใช้ 10-fold cross-validation ช่วยในการวัดผลความแม่นยำของโมเดลได้อย่างเสถียร และลดการเกิด overfitting ทำให้ได้ผลลัพธ์ที่เชื่อถือได้

3.โปรแกรม

GITHUB : https://github.com/Pattharajrin/261456_CI_Assignment3_Y3-1.git

```python
import numpy as np
import random
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# ฟังก์ชันสำหรับโหลดและเตรียมข้อมูลจากชุดข้อมูล WDBC
def load_wdbc_data(filepath):
    dataset = []
    with open(filepath, 'r') as file:
        for line in file:
            elements = line.strip().split(',')
            features = list(map(float, elements[2:]))  # ฟีเจอร์จาก index 3 ถึง 32
            label = 1 if elements[1] == 'M' else 0  # การวินิจฉัย: M = 1, B = 0
            dataset.append((features, label))
    return dataset

# ฟังก์ชันสำหรับการปกติฟีเจอร์
def normalize_features(data):
    X = np.array([item[0] for item in data])
    y = np.array([item[1] for item in data])
    X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
    return X, y

# ฟังก์ชันสำหรับแบ่งข้อมูลเป็นชุดฝึกและชุดทดสอบ
def split_into_folds(data, k=10):
    random.shuffle(data)
    fold_size = len(data) // k
    return [data[i * fold_size:(i + 1) * fold_size] for i in range(k)]

# ฟังก์ชัน Sigmoid
def sigmoid_function(x):
    return 1 / (1 + np.exp(-x))

# ฟังก์ชัน Softmax สำหรับเลเยอร์เอาท์พุท
def softmax_function(x):
    exp_scores = np.exp(x - np.max(x, axis=1, keepdims=True))
    return exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

# คลาส Multilayer Perceptron
class MLPModel:
    def __init__(self, input_dim, hidden_layers, output_dim):
        self.input_dim = input_dim
        self.hidden_layers = hidden_layers
        self.output_dim = output_dim
        self.layer_sizes = [input_dim] + hidden_layers + [output_dim]
        self.weights = [np.random.randn(self.layer_sizes[i], self.layer_sizes[i + 1]) * 0.1 for i in range(len(self.layer_sizes) - 1)]
        self.biases = [np.random.randn(1, self.layer_sizes[i + 1]) * 0.1 for i in range(len(self.layer_sizes) - 1)]

    def forward_pass(self, X):
        self.activations = [X]
        for i in range(len(self.weights) - 1):
            z = np.dot(self.activations[-1], self.weights[i]) + self.biases[i]
            self.activations.append(sigmoid_function(z))
        z = np.dot(self.activations[-1], self.weights[-1]) + self.biases[-1]
        self.output_layer = softmax_function(z)
        return self.output_layer

    def make_prediction(self, X):
        probabilities = self.forward_pass(X)
        return np.argmax(probabilities, axis=1)
```

```python
# คลาส Genetic Algorithm
class GeneticOptimizer:
    def __init__(self, pop_size, mutation_prob, num_generations):
        self.pop_size = pop_size
        self.mutation_prob = mutation_prob
        self.num_generations = num_generations
        self.fitness_history = []

    def create_initial_population(self, input_dim, hidden_layers, output_dim):
        return [MLPModel(input_dim, hidden_layers, output_dim) for _ in range(self.pop_size)]

    def perform_crossover(self, parent1, parent2):
        child = MLPModel(parent1.input_dim, parent1.hidden_layers, parent1.output_dim)
        for i in range(len(parent1.weights)):
            mask = np.random.rand(*parent1.weights[i].shape) > 0.5
            child.weights[i] = np.where(mask, parent1.weights[i], parent2.weights[i])
        return child

    def apply_mutation(self, model):
        for i in range(len(model.weights)):
            if np.random.rand() < self.mutation_prob:
                model.weights[i] += np.random.randn(*model.weights[i].shape) * 0.1

    def calculate_fitness(self, model, X, y):
        predictions = model.make_prediction(X)
        return np.mean(predictions == y)

    def evolve_population(self, X_train, y_train, X_val, y_val):
        population = self.create_initial_population(X_train.shape[1], [128], 2)
        for generation in range(self.num_generations):
            population.sort(key=lambda mlp: self.calculate_fitness(mlp, X_val, y_val), reverse=True)
            best_score = self.calculate_fitness(population[0], X_val, y_val)
            self.fitness_history.append(best_score)
            print(f"Generation {generation + 1}/{self.num_generations}, Best fitness: {best_score:.4f}")
            new_population = population[:self.pop_size // 2]  # การคัดเลือก
            for _ in range(self.pop_size // 2):  # การผสมพันธุ์
                parent1, parent2 = random.sample(new_population, 2)
                child = self.perform_crossover(parent1, parent2)
                new_population.append(child)
            population = new_population
            for model in population:  # การกลายพันธุ์
                self.apply_mutation(model)
        return population[0]

    def visualize_progress(self):
        plt.plot(self.fitness_history)
        plt.xlabel('Generation')
        plt.ylabel('Best Fitness')
        plt.title('Genetic Algorithm Optimization Progress')
        plt.show()
```

```python
# ฟังก์ชันสำหรับ k-fold cross-validation
def k_fold_validation(data, k=10):
    folds = split_into_folds(data, k)
    accuracy_scores = []
    for i in range(k):
        validation_fold = folds[i]
        training_folds = [sample for j in range(k) if j != i for sample in folds[j]]
        X_train, y_train = normalize_features(training_folds)
        X_val, y_val = normalize_features(validation_fold)

        # ใช้ Genetic Algorithm กับ MLP
        ga_optimizer = GeneticOptimizer(pop_size=20, mutation_prob=0.01, num_generations=100)
        best_model = ga_optimizer.evolve_population(X_train, y_train, X_val, y_val)

        # ทดสอบโมเดลบนชุดทดสอบ
        accuracy = ga_optimizer.calculate_fitness(best_model, X_val, y_val)
        accuracy_scores.append(accuracy)

    avg_accuracy = np.mean(accuracy_scores)
    print("Cross-validation accuracies:", accuracy_scores)
    print("Average accuracy:", avg_accuracy)

    return accuracy_scores, avg_accuracy, ga_optimizer, best_model

# ฟังก์ชันสำหรับการแสดงผลความแม่นยำ
def display_accuracy_with_average(accuracies, avg_accuracy):
    folds = range(1, len(accuracies) + 1)
    plt.bar(folds, accuracies, label='Accuracy per Fold')
    plt.axhline(avg_accuracy, color='r', linestyle='--', label=f'Average Accuracy: {avg_accuracy:.4f}')
    plt.xlabel('Fold')
    plt.ylabel('Accuracy')
    plt.title('K-Fold Cross-Validation Accuracies with Average')
    plt.legend()
    plt.show()

# ฟังก์ชันสำหรับการแสดงผล confusion matrix
def display_confusion_matrix(model, X_train, y_train):
    predictions = model.make_prediction(X_train)

    cm = confusion_matrix(y_train, predictions)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])  # Benign (0) and Malignant (1)
    disp.plot()
    plt.title('Confusion Matrix')
    plt.show()

# การทำงานหลัก
if __name__ == '__main__':
    filepath = 'wdbc.data.txt'
    data = load_wdbc_data(filepath)
    accuracy_scores, avg_accuracy, ga_optimizer, best_model = k_fold_validation(data)

    # แสดงผลค่าความแม่นยำและค่าเฉลี่ย
    display_accuracy_with_average(accuracy_scores, avg_accuracy)

    # แสดงผล confusion matrix
    X_full, y_full = normalize_features(data)  # ใช้ข้อมูลทั้งหมดสำหรับ confusion matrix
    display_confusion_matrix(best_model, X_full, y_full)

    # แสดงผลพัฒนาการ GA
    ga_optimizer.visualize_progress()
```