

**인공지능(03)**

**2023년 가을학기**

**인공지능 텀프로젝트**

**제출일: 2023.12.04**

**전공: 게임공학과**

**학번: 2019184020**

**성명: 윤은지**

**목차**

1. K-평균 클러스터링(K-means clustering)
2. 유전 알고리즘(숫자 추측하기)
3. CNN을 사용한 영상인식
4. 강화학습
5. **K-평균 클러스터링(K-means clustering)**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from matplotlib import font\_manager, rc

# 한글 폰트 설정 (Windows 기준, MacOS나 Linux에서는 해당 부분을 각 환경에 맞게 수정)

font\_path = "C:/Windows/Fonts/malgun.ttf"

font\_name = font\_manager.FontProperties(fname=font\_path).get\_name()

rc('font', family=font\_name)

# 1) NLPRK\_STA.csv 파일 읽기

data = pd.read\_csv('NLPRK\_STA.csv', encoding='euc-kr')

# 2) 국립공원 육지면적, 탐방객수 컬럼만 선택

X = data[['육지면적', '탐방객수']]

# 3) x축은 '육지면적', y축은 '탐방객수'로 설정하고 22개 국립공원을 mapping

plt.figure(figsize=(10, 6))

plt.scatter(X['육지면적'], X['탐방객수'], c='blue', label='국립공원')

plt.title('국립공원 육지면적 vs 탐방객수')

plt.xlabel('육지면적')

plt.ylabel('탐방객수')

# 4) K-평균 클러스터링 적용 (k=3)

kmeans = KMeans(n\_clusters=3, max\_iter=100, random\_state=42)

X['Cluster'] = kmeans.fit\_predict(X)

# 클러스터 중심 표시

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='X', label='클러스터 중심')

# 결과 그래프 표시

plt.legend()

plt.show()

1. NLPRK\_STA.csv 파일 읽기

-‘pd.read\_csv’ 함수를 사용하여 ‘NLPRK\_STA.csv’ 파일을 읽었습니다.

-‘euc-kr’은 한글 인코딩을 사용하기 위한 설정입니다.

1. X = data[[ ‘육지면적’ , ‘탐방객수’]]

-데이터 프레임에서 ‘육지면적’과 ‘탐방객수’열만 선택하여 ‘X’에 저장합니다.

1. Matplotlib을 사용하여 국립공원의 육지면적과 탐방객수를 산점도로 나타냅니다.

4) KMeans 모델을 생성하고, 국립공원 데이터를 3개의 클러스터로 클러스터링 했습니다.

-fit\_predict를 사용하여 각 데이터 포인트가 속하는 클러스터를 예측하고, 이를 Cluster열로 추가합니다.

1. 각 클러스터의 중심을 빨간색으로 표시합니다.
2. 최종적으로 그래프를 표시합니다.
3. 실행 화면

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

이를 통해 국립공원 데이터를 K-평균 클러스터링을 사용하여 3개의 클러스터로 나눈 후, 각 클러스터의 중심을 시각화한 결과를 얻을 수 있습니다. 또한 국립공원을 육지면적과 탐방객수에 따라 유형 분류한 것을 확인할 수 있습니다.

1. **유전 알고리즘(숫자 추측하기)**

import random

# 게임 규칙 설정

N = 6  # 염색체 길이 (숫자야구 숫자의 자릿수)

MAX\_GENERATIONS = 100  # 최대 세대 수

POPULATION\_SIZE = 10  # 염색체 개수

def generate\_chromosome():

    """랜덤한 N자리 숫자로 염색체 생성"""

    return [random.randint(0, 9) for \_ in range(N)]

def calculate\_fitness(solution, guess):

    """적합도 계산: 스트라이크를 5점, 볼을 1점으로 환산"""

    strikes = sum(s == g for s, g in zip(solution, guess))

    balls = sum(s in guess and s != g for s, g in zip(solution, guess))

    return strikes \* 5 + balls

def select\_parents(population, fitness\_scores):

    """룰렛 방식으로 부모 선택"""

    total\_fitness = sum(fitness\_scores)

    selection\_probabilities = [fitness / total\_fitness for fitness in fitness\_scores]

    parents = random.choices(population, weights=selection\_probabilities, k=2)

    return parents

def crossover(parent1, parent2):

    """교차 연산을 통해 자손 생성"""

    crossover\_point = random.randint(1, N - 1)

    child = parent1[:crossover\_point] + parent2[crossover\_point:]

    return child

def mutate(child):

    """돌연변이 연산을 통해 자손의 일부를 변형"""

    mutation\_point = random.randint(0, N - 1)

    child[mutation\_point] = random.randint(0, 9)

    return child

# 초기 염색체 생성

population = [generate\_chromosome() for \_ in range(POPULATION\_SIZE)]

for generation in range(MAX\_GENERATIONS):

    # 적합도 계산 및 출력

    fitness\_scores = [calculate\_fitness(solution=[1, 2, 3, 4, 5, 6], guess=chromosome) for chromosome in population]

    best\_chromosome = population[fitness\_scores.index(max(fitness\_scores))]

    print(f"세대 {generation + 1}, 최적 염색체: {best\_chromosome}, 적합도: {max(fitness\_scores)}")

    # 다음 세대를 위해 새로운 염색체 생성

    new\_population = []

    for \_ in range(POPULATION\_SIZE // 2):

        # 부모 선택

        parent1, parent2 = select\_parents(population, fitness\_scores)

        # 교차 연산을 통해 자손 생성

        child1 = crossover(parent1, parent2)

        child2 = crossover(parent2, parent1)

        # 돌연변이 연산

        child1 = mutate(child1)

        child2 = mutate(child2)

        new\_population.extend([child1, child2])

    # 새로 생성된 자손으로 염색체 갱신

    population = new\_population

# 최종 결과 출력

best\_solution = population[fitness\_scores.index(max(fitness\_scores))]

print(f"최적 해: {best\_solution}")

1. Generate\_chromosome 함수 :

-염색체를 생성하는 함수입니다.

-‘N’ 자릿수의 랜덤한 숫자를 생성하여 염색체로 반환합니다.

1. Calculate\_fitness 함수 :

-염색체의 적합도를 계산하는 함수입니다. 여기서는 스트라이크를 5점, 볼을 1점으로 환산합니다.

-‘solution’ : 정답 염색체(숫자야구의 정답)

-‘guess’ : 추측한 염색체(유전 알고리즘으로 생성된 염색체)

1. Select\_parents 함수 :

-룰렛 방식을 사용하여 부모 염색체를 선택하는 함수입니다. 적합도에 비례한 확률로 선택됩니다.

-‘population’ : 현재 세대의 염색체들

-‘fitness\_scores’ : 각 염색체의 적합도 점수

1. Crossover 함수 :

-두 부모 염색체를 교차 연산을 통해 자손 염색체를 생성하는 함수입니다.

-‘parent1’, ‘parent2’ : 선택된 부모 염색체

1. Mutate 함수 :

-돌연변이 연산을 통해 자손의 일부를 변형하는 함수입니다.

-‘child’ : 돌연변이를 적용할 자손 염색체

1. 실행 화면

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

위 함수들은 유전 알고리즘의 기본 연산들로, 이를 활용하여 세대를 진행하고 최적의 해를 찾아가는 과정이 반복됩니다. 코드에서는 주어진 규칙에 따라 숫자 야구 게임의 최적해를 찾기 위해 염색체를 생성하고, 적합도를 계산하고, 교차와 돌연변이를 통해 다음 세대의 염색체를 생성했습니다.

1. **CNN을 사용한 영상인식**

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

# CIFAR-10 데이터 로드

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()

# 픽셀 값을 0~1 사이로 정규화

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

# CNN 모델 구축

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(10, activation='softmax'))

# 모델 컴파일

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

# 모델 훈련

model.fit(train\_images, train\_labels, epochs=10)

# 테스트 이미지 중에서 일부를 사용하여 예측

predictions = model.predict(test\_images[:5])

# 예측 결과 출력

for i in range(5):

    predicted\_label = tf.argmax(predictions[i])

    true\_label = test\_labels[i][0]

    print(f"Predicted: {predicted\_label.numpy()}, True: {true\_label}")

# 검증 결과 시각화

fig, axes = plt.subplots(1, 5, figsize=(15, 3))

for i in range(5):

    axes[i].imshow(test\_images[i])

    axes[i].set\_title(f"Predicted: {tf.argmax(predictions[i])}")

    axes[i].axis('off')

plt.show()

1. CIFAR-10 데이터 로드 :

CIFAR-10 데이터셋을 로드합니다. 이 데이터셋은 60000개의 32X32 컬러 이미지로 구성되어 있습니다.

1. 픽셀 값 정규화 :

픽셀 값을 [0, 1] 범위로 정규화합니다.

1. CNN 모델 구축 :

-CNN 모델을 구성합니다. 합성곱 레이어, 맥스 풀링 레이어 및 밀집 레이어로 이루어져 있습니다.

-Adam 옵티마이저를 사용하고 손실 함수로는 희소 범주형 크로스 엔트로피를 사용하여 모델을 컴파일합니다.

1. 모델 훈련 :

모델을 훈련합니다.

1. 모델 예측 및 시각화 :

테스트 이미지 중에서 일부를 사용하여 예측하고 결과를 출력하며 시각화했습니다.

TensorFlow와 Keras를 사용하여 CIFAR-10 데이터셋에서 이미지 분류를 위한 합성곱 신경망을 구현했습니다.

**4. 강화학습**

import numpy as np

class QLearningAgent:

    def \_\_init\_\_(self, num\_states, num\_actions, step\_size=0.01, discount\_factor=0.9, epsilon=0.9):

        self.num\_states = num\_states

        self.num\_actions = num\_actions

        self.q\_table = np.zeros((num\_states, num\_actions))  # Q-Table 초기화

        self.step\_size = step\_size

        self.discount\_factor = discount\_factor

        self.epsilon = epsilon

    def select\_action(self, state):

        # Epsilon-Greedy 정책을 사용하여 액션 선택

        if np.random.rand() < self.epsilon:

            return np.argmax(self.q\_table[state])

        else:

            return np.random.choice(self.num\_actions)

    def update\_q\_table(self, state, action, reward, next\_state):

        # 벨만 최적 방정식을 사용한 Q-Table 업데이트

        q\_1 = self.q\_table[state][action]

        q\_2 = reward + self.discount\_factor \* np.max(self.q\_table[next\_state])

        self.q\_table[state][action] += self.step\_size \* (q\_2 - q\_1)

# 간단한 환경 예제

num\_states = 16

num\_actions = 4

# Q-Learning 에이전트 초기화

agent = QLearningAgent(num\_states=num\_states, num\_actions=num\_actions)

# 학습 예제

num\_episodes = 1000

for episode in range(num\_episodes):

    state = 0  # 시작 상태

    total\_reward = 0

    while state != num\_states - 1:  # 목표 상태에 도달할 때까지 반복

        action = agent.select\_action(state)

        # 다음 상태로 이동 (간단한 예제에서는 상태 및 보상이 간소화되어 있음)

        next\_state = min(state + 1, num\_states - 1)

        # 보상 및 Q-Table 업데이트

        reward = 0 if next\_state != num\_states - 1 else 100  # 목표 도달 시 높은 보상

        agent.update\_q\_table(state, action, reward, next\_state)

        total\_reward += reward

        state = next\_state

    if episode % 100 == 0:

        print(f"에피소드: {episode}, 총 보상: {total\_reward}")

# 학습된 Q-Table 출력

print("최종 Q-Table:")

print(agent.q\_table)

1. QLearningAgent 클래스 :

-‘\_\_init\_\_’ : 초기화 함수로, Q-Learning 에이전트 속성들을 설정합니다.

-‘select\_action’ : Epsilon-Greedy 정책에 따라 액션을 선택합니다.

-‘update\_q\_table’ : 벨만 최적 방정식을 사용하여 Q-Table을 업데이트합니다.

1. 간단한 환경 예제 :

-‘num\_states’ : 상태의 수로, 16개의 상태가 있습니다.

-‘num\_actions’ : 가능한 액션의 수로, 4개의 액션이 있습니다.

1. Q-Learning 에이전트 초기화 :

agent = QLearningAgent(num\_states=num\_states, num\_actions=num\_actions)

Q-Learning 에이전트를 초기화합니다.

1. 학습 예제 :

-‘num\_episodes’ : 학습에 사용할 에피소드 수입니다.

-각 에피소드마다 에이전트는 시작 상태부터 목표 상태에 도달할 때까지 액션을 선택하고 Q-Table을 업데이트합니다.

1. 에피소드 루프 :

for episode in range(num\_episodes):

각 에피소드에서 시작 상태부터 목표 상태에 도달할 때까지의 루프입니다.

1. 에피소드 내용 :

-‘while status != num\_status – 1 : ‘ : 목표 상태에 도달할 때까지 반복합니다.

-‘action = agent.select\_action(status)’ : Epsilon-Greedy 정책을 사용하여 액션을 선택합니다.

-‘next\_status = min(state + 1, num\_status – 1)’ : 다음 상태로 이동합니다.

-‘reward’ : 보상을 설정합니다. 목표 도달 시 높은 보상을 주어 학습을 유도합니다.

-‘agent.update\_q\_table(state, action, reward, next\_state)’ : Q-Table을 업데이트합니다.

1. 학습된 Q-Table 출력 :

학습이 완료된 후 최종 Q-Table을 출력합니다. 이 테이블은 각 상태 및 액션에 대한 가치를 나타냅니다.

1. 실행 화면
2. 텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

   자동 생성된 설명

Q-Learning 에이전트를 구현하고, 이를 간단한 환경에서 학습시키는 코드를 구현했습니다.