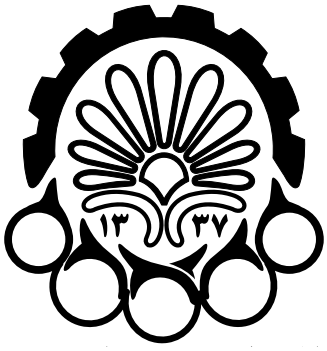

دانشگاه صنعتي اميركبير

(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

گزارش ششم درس هوش مصنوعی

نگارش

آرمان صالحی

استاد راهنما

دکتر مهدی قطعی

بهار 1403

چکیده

الگوریتم ژنتیک یک روش بهینه‌سازی الهام گرفته از طبیعت است که شامل مراحل انتخاب، ترکیب و جهش برای بهبود راه‌حل‌ها است. در این کد، ما از الگوریتم ژنتیک برای خوشه‌بندی داده‌ها استفاده کرده‌ایم و از معیار silhouette_score برای ارزیابی کیفیت خوشه‌بندی بهره برده‌ایم. با تکرار این مراحل، الگوریتم به تدریج راه‌حل‌های بهتری را پیدا می‌کند.

واژه‌های کلیدی:

Genetic Algorithm – Data Frame – Label Encoder – Fit – StandardScaler - silhouette_score - Mutation

فهرست مطالب **صفحه**

گزارش کار

1. وارد کردن کتابخانه‌ها

```python
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import silhouette_score
```

2. خواندن دیتاست از فایل اکسل

```python
# Reading and processing data from dataset
data = pd.read_excel('Dry_Bean_Dataset.xlsx')
```

این خط فایل اکسل را با استفاده از pandas می‌خواند و آن را به یک DataFrame تبدیل می‌کند.

3. پردازش پیش از داده

```python
data = data.dropna()
labelEncoder = LabelEncoder()
data['Class'] = labelEncoder.fit_transform(data['Class'])
```

- data.dropna(): حذف ردیف‌های حاوی مقادیر گمشده.

- LabelEncoder: برای تبدیل برچسب‌های کلاس به مقادیر عددی استفاده می‌شود.

- data['Class'] = label_encoder.fit_transform(data['Class']): تبدیل برچسب‌های کلاس به مقادیر عددی و جایگزینی در DataFrame.

4. جداسازی ویژگی‌ها و Labelها

```
13  X = data.drop('Class', axis=1)
14  y = data['Class']
```

- X: شامل ویژگی‌های داده‌ها (بدون ستون برچسب).

- y: شامل برچسب‌های کلاس.

5. نرمال سازی داده‌ها

```
16  scaler = StandardScaler()
17  X = scaler.fit_transform(X)
```

- StandardScaler: برای نرمال‌سازی داده‌ها به طوری که میانگین صفر و واریانس یک داشته باشند.

- X = scaler.fit_transform(X): نرمال‌سازی داده‌های ویژگی.

6. تعریف توابع الگوریتم ژنتیک

a. مقداردهی اولیه جمعیت

```
20  #Implementation of genetic algorithm
21  def initializePopulation(size, numFeatures):
22      return np.random.rand(size, numFeatures)
```

- این تابع یک جمعیت اولیه از راه‌حل‌های تصادفی ایجاد می‌کند. اندازه جمعیت و تعداد ویژگی‌ها به عنوان ورودی دریافت می‌شود.

b. محاسبه شایستگی

```
24  def calculateFitness(population, X, y):
25      fitness = []
26      for individual in population:
27          distances = np.linalg.norm(X[:, np.newaxis] - individual, axis=2)
28          labels = np.argmin(distances, axis=1)
29          if len(np.unique(labels)) > 1:
30              fitness.append(silhouette_score(X, labels))
31          else:
32              fitness.append(0)
33      return np.array(fitness)
```

- این تابع شایستگی هر فرد در جمعیت را محاسبه می‌کند.

- distances: محاسبه فاصله هر نقطه داده با مراکز خوشه‌های کاندید.

- labels: اختصاص هر نقطه به نزدیک‌ترین مرکز خوشه.

- silhouette_score: ارزیابی کیفیت خوشه‌بندی. اگر تعداد خوشه‌ها کمتر از دو باشد، شایستگی صفر است.

c. انتخاب

```
35  def selection(population, fitness):
36      indices = np.argsort(fitness)[-len(population)//2:]
37      return population[indices]
```

- این تابع افراد با شایستگی بالاتر را برای تولید نسل بعدی انتخاب می‌کند. نیمی از افراد با بالاترین شایستگی انتخاب می‌شوند.

d. ترکیب (Crossover)

```
39  def crossover(parent1, parent2):
40      crossoverPoint = np.random.randint(0, len(parent1))
41      child1 = np.concatenate([parent1[:crossoverPoint], parent2[crossoverPoint:]])
42      child2 = np.concatenate([parent2[:crossoverPoint], parent1[crossoverPoint:]])
43      return child1, child2
```

- این تابع دو والد را ترکیب می‌کند تا دو فرزند تولید کند. نقطه ترکیب به صورت تصادفی انتخاب می‌شود و بخش‌های والدین تعویض می‌شوند.

e. جهش (Mutation)

```python
def mutation(child, mutationRate):
    for i in range(len(child)):
        if np.random.rand() < mutationRate:
            child[i] = np.random.rand()
    return child
```

- این تابع تغییرات تصادفی کوچکی در ژن‌های فرزند ایجاد می‌کند. احتمال جهش توسط mutation_rat تعیین می‌شود.

7. پارامترهای الگوریتم ژنتیک

```python
populationSize = 50
numGenerations = 100
mutationRate = 0.1
numFeatures = X.shape[1]
```

- population_size: تعداد افراد در جمعیت.

- num_generations: تعداد نسل‌ها.

- mutation_rate: نرخ جهش.

- num_features: تعداد ویژگی‌ها (طول کروموزوم‌ها).

8. مقداردهی اولیه جمعیت

```python
population = initializePopulation(populationSize, numFeatures)
```

- تولید جمعیت اولیه از راه‌حل‌های تصادفی.

9. اجرای الگوریتم ژنتیک

```
58  for generation in range(numGenerations):
59      fitness = calculateFitness(population, X, y)
60      selected_population = selection(population, fitness)
61      nextPopulation = []
62
63      while len(nextPopulation) < populationSize:
64          parents = np.random.choice(range(len(selected_population)), size=2, replace=False)
65          parent1, parent2 = selected_population[parents[0]], selected_population[parents[1]]
66          child1, child2 = crossover(parent1, parent2)
67          child1 = mutation(child1, mutationRate)
68          child2 = mutation(child2, mutationRate)
69          nextPopulation.extend([child1, child2])
70
71      population = np.array(nextPopulation)
```

- برای هر نسل:

- calculate_fitness: محاسبه شایستگی جمعیت.

- selection: انتخاب افراد با شایستگی بالاتر.

تولید نسل بعدی: ترکیب و جهش والدین انتخاب شده برای تولید فرزندان.

10. انتخاب بهترین فرد

```
73  bestIndividual = population[np.argmax(calculateFitness(population, X, y))]
74  print(bestIndividual)
```

- np.argmax(calculate_fitness(population, X, y)): پیدا کردن بهترین فرد در جمعیت.