به نام خدا

گزارش پروژه سوم - ژنتیک

نیایش خانی ۹۹۵۲۱۲۳۵

قسمت اول : خوشه بندی

ابتدا کتابخانه های لازم را import می کنیم. سپس کلاس کروموزوم را پیاده سازی می‌کنیم.

```
class Chromosome:
    def __init__(self, genes, length):
        self.genes = genes
        self.length = length
        self.fitness = 0

    def randomGenerateChromosome(self):
        for i in range(0, self.length):
            gen = float('%.2f' % random.uniform(0.0, 1.0))
            self.genes.append(gen)

        return self

    def mutate(self, mutation_rate):
        for i in range(self.length):
            if random.random() < mutation_rate:
                self.genes[i] = float('%.2f' % random.uniform(0.0, 1.0))
```

این کلاس نشان دهنده یک فرد در جمعیت و در حقیقت یک کروموزوم در الگوریتم ژنتیک است که هر کروموزوم حاوی ژن هایی است که مرکزهای cluster ها را نشان می دهد. با استفاده از randomGenerateChromosome یک کروموزوم با ژن های تصادفی را مقداردهی اولیه می کنیم. همچنین برای اعمال mutation از فانکشن mutate استفاده شده است.

با کمی جستجو عملیاتی برای mutation پیدا کردم که به این صورت کار می کند : ابتدا به هر ژن مقدار تصادفی بین ۰ و ۱ اختصاص داده می شود. سپس عملیات جهش به طور تصادفی یک ژن را در یک کروموزوم با احتمال مشخصی (نرخ جهش) تغییر می دهد به این صورت که برای هر ژن در کروموزوم، بررسی می شود که آیا باید بر اساس میزان جهش، جهش یابد یا خیر. اگر عدد تصادفی تولید شده کمتر از میزان جهش باشد، ژن جهش یافته است. مقدار ژن جهش یافته با یک مقدار تصادفی جدید بین ۰ و ۱ جایگزین می شود.

کلاس تعریف شده بعدی Cluster می باشد که به این صورت پیاده سازی شده است :

```python
class Cluster:
    def __init__(self, centroid):
        self.centroid = centroid
        self.points = []

    def computeS(self):
        n = len(self.points)
        if n == 0:
            return 0
        s = np.sum(np.linalg.norm(self.points - self.centroid, axis=1))
        return s / n
```

کلاس Cluster نشان دهنده یک خوشه است. هر خوشه دارای یک مرکز (میانگین نقاط اختصاص داده شده به آن) و لیستی از نقاط اختصاص داده شده به آن است. روش computeS میانگین فاصله (های) نقاط در خوشه را از مرکز آن محاسبه می کند.

سپس به سراغ کلاس Clustering می رویم. (به دلیل طولانی بودن کد، قرار داده نشده است.)

کلاس Clustering مسئله خوشه بندی را نشان می دهد. داده های ورودی (مجموعه داده) و حداکثر تعداد خوشه kmax را به عنوان ورودی می گیرد.

- initialize_clusters : با انتخاب تصادفی نقاط داده kmax از داده های ورودی به عنوان مرکز اولیه، خوشه ها را مقداردهی اولیه می کند. سپس لیستی از آبجکت های Cluster را برمی‌گرداند که هر کدام حاوی یک centroid با یک نقطه انتخابی تصادفی است.

- assign_points_to_clusters : هر نقطه دیتا را به خوشه ای با نزدیکترین مرکز اختصاص می دهد. روی تمام نقاط دیتا تکرار می شود و فاصله آن تا هر مرکز خوشه را با استفاده از فاصله اقلیدسی محاسبه می کند. اگر فاصله محاسبه شده (dist) کمتر از حداقل فاصله فعلی (min_dist) باشد، حداقل فاصله را به روز می کند و نقطه داده را به خوشه ای با نزدیکترین مرکز (closest_cluster) اختصاص می دهد. به این وسیله Objective function که minimize کردن فاصله اقلیدوسی بود، پیاده سازی می شود.

- update_cluster_centroids : مرکزهای خوشه ها را بر اساس میانگین نقاط داده اختصاص داده شده به هر خوشه به روز می کند. در هر خوشه تکرار می شود و میانگین تمام نقاط داده اختصاص داده شده به آن خوشه را محاسبه می کند. مرکز خوشه به میانگین محاسبه شده آپدیت می شود.

- euclidean_distance : این تابع فاصله اقلیدسی بین دو نقطه را محاسبه می کند. از تابع "numpy.linalg.norm" برای محاسبه طول بردار از "point1" تا "point2" استفاده می کند.

- k_means_objective : این تابع objective function است که الگوریتم k-means را برای خوشه های داده شده محاسبه می کند. تابع هدف مجموع فواصل اقلیدوسی همه نقاط تا مرکز خوشه مربوطه آنهاست. هدف الگوریتم k-means به حداقل رساندن این تابع هدف است.

حال به سراغ نرمالیزیشن می رویم.

```
def minmax_normalize(data):
    min_vals = np.min(data, axis=0)
    max_vals = np.max(data, axis=0)
    normalized_data = (data - min_vals) / (max_vals - min_vals)
    return normalized_data
```

روش استفاده شده در این تابع نرمال سازی min-max است. نرمال سازی حداقل حداکثر تکنیکی است که اغلب برای مقیاس خطی هر ویژگی (به عنوان مثال، ستون) در داده ها به محدوده مشترک [۰، ۱] استفاده می شود. این کار را با کم کردن حداقل مقدار هر ویژگی از هر نقطه داده در آن ویژگی، و سپس تقسیم بر دامنه مقادیر (حداکثر - حداقل) برای آن ویژگی انجام می دهد. در آخر نتیجه ریترن شده، یک مجموعه داده است که در آن هر ویژگی دارای مقادیری از ۰ تا ۱ است.

در آخر برای اجرای کل ببرنامه از تابع main استفاده شده است.

```
def main():
    iris = load_iris()
    data = iris.data
    data = minmax_normalize(data)

    clustering = Clustering(data, kmax=3)
    best_clusters = clustering.run_genetic_algorithm(
        num_individuals=50, max_generations=100, mutation_rate=0.01,
crossover_rate=0.8)

    for i, cluster in enumerate(best_clusters):
        print(f"Cluster {i+1} Centroid: {cluster.centroid}"
```

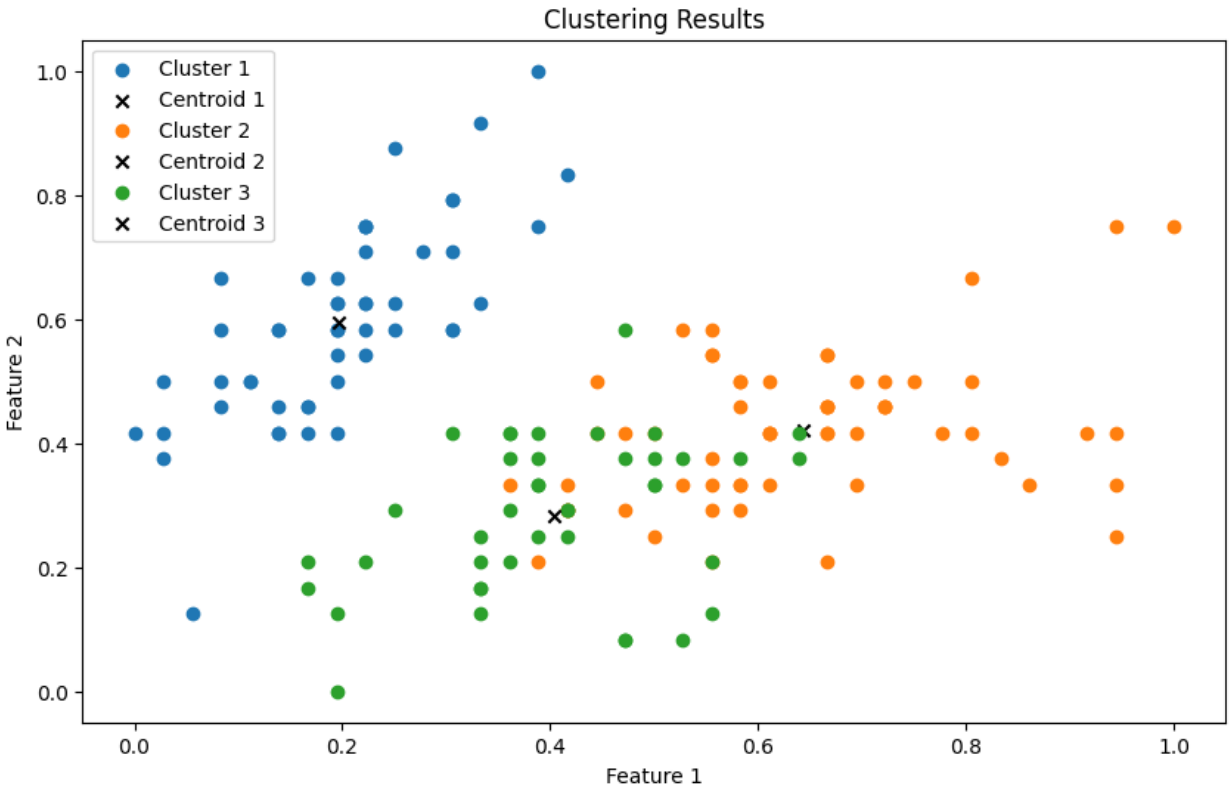
ابتدا دیتاست Iris را با استفاده از تابع "load_iris()" از کتابخانه "sklearn.datasets" لود می کنیم. سپس دیتا با استفاده از تابع 'minmax_normalize(data)'نرمال می شوند که مطمئن شویم همه ویژگی‌های داده‌ها مقیاس یکسانی دارند. سپس یک آبجکت از کلاس 'Clustering' با داده های نرمال شده و حداکثر تعداد مشخصی از خوشه ها ('kmax=3') ایجاد می کنیم.
الگوریتم ژنتیک با فراخوانی روش "run_genetic_algorithm" روی داده ها اجرا می شود. پارامترهای الگوریتم ژنتیک در اینجا با انجام آزمون و خطا مشخص شده اند و سعی کردم از اعدادی که رایج هستند استفاده کنم.(تعداد افراد در جمعیت («تعداد_افراد=۵۰»)، حداکثر تعداد نسل ها («حداکثر_نسل=۱۰۰»)، میزان mutation (mutation_rate=0.01)، و نرخ crossover (crossover_rate=0.8)). سپس مرکز بهترین خوشه های یافت شده توسط الگوریتم ژنتیک چاپ می شود.
در نهایت نتایج clustering با استفاده از matplotlib ترسیم می شود. هر خوشه با رنگ متفاوتی نشان داده می شود و مرکزها با علامت "x" مشخص می شوند.

خروجی کد :

```
Cluster 1 Centroid: [0.19611111 0.595      0.07830508 0.06083333]
Cluster 2 Centroid: [0.64359699 0.42019774 0.75064637 0.77048023]
Cluster 3 Centroid: [0.40311653 0.28150407 0.5345184  0.49288618]
```

همانطور که مشاهده می شود سه cluster داریم و centroid آنها مشخص شده است.

با plot کردن نتیجه به شکل زیر می‌رسیم.



نمودار سه کلاستر از نقاط داده را نشان می دهد که هر کدام با رنگ متفاوتی مشخص شده اند: آبی برای کلاستر ۱، نارنجی برای کلاستر ۲، و سبز برای کلاستر ۳ . هر نقطه داده در نمودار نشان دهنده یک دیتا در مجموعه داده است. علامت های "X" نیز مرکز کلاستر ها را نشان می دهند.

دلیل استفاده از الگوریتم k-means در کلاسترینگ : K-means می تواند در بهینه محلی گیر کند و به انتخاب اولیه مرکزها حساس است. از سوی دیگر، کلاسترینگ با استفاده از الگوریتم‌های تکاملی اغلب می‌تواند راه‌حل‌های بهتری پیدا کند و حساسیت کمتری به مقداردهی اولیه دارد، اما باید در نظر داشت که پیچیدگی محاسباتی بیشتری دارد.

قسمت دوم : گراف

در این قسمت، ما از الگوریتم ژنتیک برای حل مسئله فروشنده دوره گرد (TSP) استفاده می کنیم. در این مسئله، یک فروشنده با لیستی از شهرها مواجه است و باید مسیری را پیدا کند که کمترین هزینه را داشته باشد، به شرطی که هر شهر را فقط یک بار بازدید کند و به محل اولیه خود برگردد.

```
# Initialize a population of paths (chromosomes)
def initialize_population(nodes,pop_size):
    max_nod_num = max(nodes)
    population = []
    for i in range(pop_size):
        chromosome = []
        # Create a fully connected path
        while len(chromosome) != len(nodes):
            rand_node = np.random.randint(max_nod_num+1)
            # Prevent repeated additions of nodes in the same chromosome
            if rand_node not in chromosome:
                chromosome.append(rand_node)
        population.append(chromosome)
    return population
```

در این فانکشن یک جمعیت اولیه از مسیرها (یا کروموزوم‌ها) را ایجاد می‌کنیم. هر مسیر یک دنباله از گره‌ها است که هر گره فقط یک بار در آن ظاهر می‌شود. این تابع تعدادی مسیر تصادفی (به اندازه pop_size) ایجاد می‌کند که هر کدام شامل تمام گره‌ها هستند.

```
# Calculate the total cost (distance) of a path
def fitness(distance_matrix, chromosome):
    total_cost = 0
    for i in range(1, len(chromosome)):
        total_cost += distance_matrix[chromosome[i-1]][chromosome[i]]
    total_cost += distance_matrix[chromosome[-1]][chromosome[0]]
    return total_cost
```

تابع سازگاری یا Fitness Function به گونه ای عمل می‌کند که هزینه کل یک مسیر داده شده (یا کروموزوم) را محاسبه می‌کند. هزینه کل برابر است با مجموع فواصل بین گره‌های متوالی در مسیر، با این فرض که مقادیر فاصله از distance_matrix بدست می‌آیند.
در مسئله فروشنده دوره‌گرد، هزینه کمتر به معنی سازگاری بیشتر است، بنابراین ما می‌خواهیم این هزینه را مینیمم کنیم.

```
# Select the best (shortest) paths from the population
def selection(parent_gen,graph_edges,elite_size):
    costs = []
    selected_parent = []
    pop_fitness = []
    for i in range(len(parent_gen)):
        costs.append(fitness(graph_edges,parent_gen[i]))
        pop_fitness.append((costs[i],parent_gen[i]))
    # Sort according to path costs
```

```
    pop_fitness.sort(key = lambda x: x[0])
    # Select only top elite_size fittest chromosomes in the population
    for i in range(elite_size):
        selected_parent.append(pop_fitness[i][1])
    return selected_parent,pop_fitness[0][0],selected_parent[0]
```

این تابع به عنوان یک عملگر انتخاب در الگوریتم ژنتیک عمل می کند که از میان جمعیت فعلی، elite_size تعداد از بهترین (کمترین هزینه) مسیرها را انتخاب می کند.

سپس crossover پیاده سازی شده است که به دلیل طولانی بودن کد در اینجا قرار نگرفته است. این تابع به گونه ای عمل می کند که دو والد (مسیر) به صورت تصادفی انتخاب می شوند و یک فرزند (مسیر جدید) با ترکیب بخش هایی از هر دو والد ایجاد می شود. این عملیات با انتخاب دو نقطه تصادفی در والدین و سپس ترکیب بخش های بین این دو نقطه از هر دو والد انجام می شود.

سپس برای mutation، یک مسیر (والد) انتخاب می شود و سپس n_mutations جفت نود در مسیر به صورت تصادفی جابجا می شوند. این عملیات باعث می شود تا تنوع در جمعیت حفظ شود و الگوریتم در حالت های بدبینانه گیر نکند.

در آخر فانکشن genetic_algorithm تابع اصلی است که الگوریتم ژنتیک را پیاده سازی می کند. جمعیتی از مسیرها را مقداردهی اولیه می کند، سپس در تعداد مشخصی از نسل ها تکرار می شود. در هر نسل، بهترین مسیرها را انتخاب می کند، آنها را برای ایجاد فرزندان پرورش می دهد و کودکان را جهش می دهد. بهترین مسیرها و بچه ها جمعیت جدید را برای نسل بعدی تشکیل می دهند. بعد از همه نسل ها بهترین مسیر و هزینه اش را برمی گرداند.

سپس یک نمونه را به عنوان ورودی قرار می‌دهیم و بقیه پارامتر ها را مانند تعداد generation ها، تعداد انتخاب بهترین ها و ... را مشخص کرده و خروجی را چاپ می کنیم.

نمونه ورودی ماتریسی است که در pdf سوالات داده شده است.

```
# Example usage:
distance_matrix = [[0, 29, 20, 21, 16],
                   [29, 0, 15, 19, 28],
                   [20, 15, 0, 13, 25],
                   [21, 19, 13, 0, 17],
                   [16, 28, 25, 17, 0]]
```

خروجی به دست آمده به ازای این ورودی :

```
The path : 0 -> 4 -> 3 -> 1 -> 2 -> 0
Total cost :  87
```

در قسمت بعدی نیز کد های لازم برای visualize کردن گراف مورد نظر و مشخص کردن مسیر بهینه پیاده سازی شده است که خروجی آن به این صورت است :