Amirkabir University of Technology
(Tehran Polytechnic)

Applied Machine Learning Course

By Dr. Nazerfard

CE5501 | Spring 2024

Assignment (5)

Name: Esmaeil Khosravi

S_ID: 402131046

Email: es.khosravi@aut.ac.ir

فهرست مطالب

سوال ۱

قسمت 1

در این قسمت، ماتریس همبستگی بین فیچرهای دیتاست با استفاده از متد **()corr** ایجاد شده است. فیچر **country** که یک فیچر غیرعددی است و اسم کشورها را نشان می دهد، از دیتاست حذف شده است زیرا که سایر فیچر ها همبستگی معناداری با اسم کشور ندارند. همچنین نمودار همبستگی بین تمام فیچر ها با استفاده از متد **pairplot** از کتابخانه **sns** رسم شده است. با توجه به ماتریس و نمودار بین بعضی از ویژگی ها همبستگی واضحی وجود دارد مانند **child_mort** و **total_ref**.

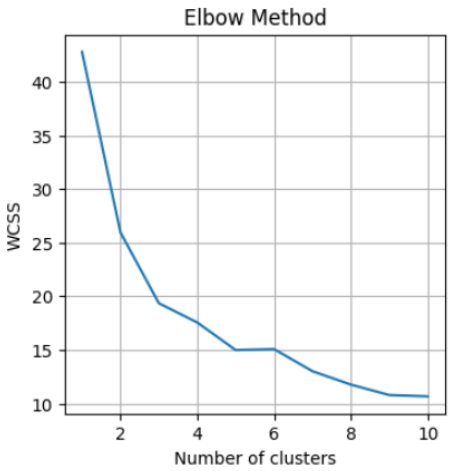
قسمت 2

در این قسمت نرمالسازی دیتاست با استفاده از ماژول **MinMaxScaler** انجام شده است که مقادیر عددی داده ها را در یک بازه بین صفر تا یک مقیاس بندی می کند. این عمل نرمالسازی به الگوریتم یادگیرنده کمک می کند تا بهتر به جواب بهینه همگرا شود.

قسمت 3

در این قسمت با استفاده از روش **Elbow Method** بهترین مقدار برای تعداد خوشه ها (پارمتر **k** در الگوریتم **Kmeans**) برای استفاده در آموزش مدل خوشه بندی، بدست آمده است. نمودار حاصل از این روش که به ازای تعداد خوشه های مختلف و مقدار **WCSS (within-cluster-**

sum-of-square) رسم شده است در ادامه نشان داده شده است که نشان می دهد بهترین مقدار می تواند عدد ۳ برای تعداد خوشه ها باشد.۰



silhouette score: یک معیار برای ارزیابی کیفیت خوشه‌بندی در تحلیل داده‌ها است. این معیار برای تعیین میزان همگنی (**homogeneity**) و تفکیک‌پذیری (**separability**) خوشه‌ها به کار می‌رود. به طور کلی، هر چه مقدار این امتیاز بالاتر باشد، خوشه‌بندی انجام‌شده بهتر است. بنابراین با این روش هم میتوان تعداد خوشه های مناسب برای مدل **kmeans** را بافت. که نتیجه استفاده از این روش در ادامه آمده است.

```
For n_clusters = 2 The average silhouette_score is : 0.37671430588173554
For n_clusters = 3 The average silhouette_score is : 0.34265474105126204
For n_clusters = 4 The average silhouette_score is : 0.29655545967060665
For n_clusters = 5 The average silhouette_score is : 0.23981104464522346
For n_clusters = 6 The average silhouette_score is : 0.2461246087924544
For n_clusters = 7 The average silhouette_score is : 0.20388129165571853
For n_clusters = 8 The average silhouette_score is : 0.2411290372608325
For n_clusters = 9 The average silhouette_score is : 0.18663659448568676
For n_clusters = 10 The average silhouette_score is : 0.18987312770029532
```

تصویر بالا نشان می دهد تعداد خوشه ۲ بیشترین امتیاز را دارد، پس طبق این روش بهترین تعداد خوشه ها عدد ۲ می باشد.

قسمت 4

در این قسمت یک مدل **kmeans** با تعداد خوشه های ۳ ایجاد شده است که وضعیت کشور ها در دبتاست را با توجه به ویژگی های آن ها در یکی از این سه دسته قرار می دهد. کشور های با وضعیت رفاهی و بهداشتی و اقتصادی بهتر در دسته ۱ قرار گرفته اند کشور های با شرایط بدتر به ترتیب در دسته های ۰و ۲ قرار گرفته اند. در نهایت عملکرد این مدل با امتیاز **silhouette** نشان داده شده است.

silhouette score

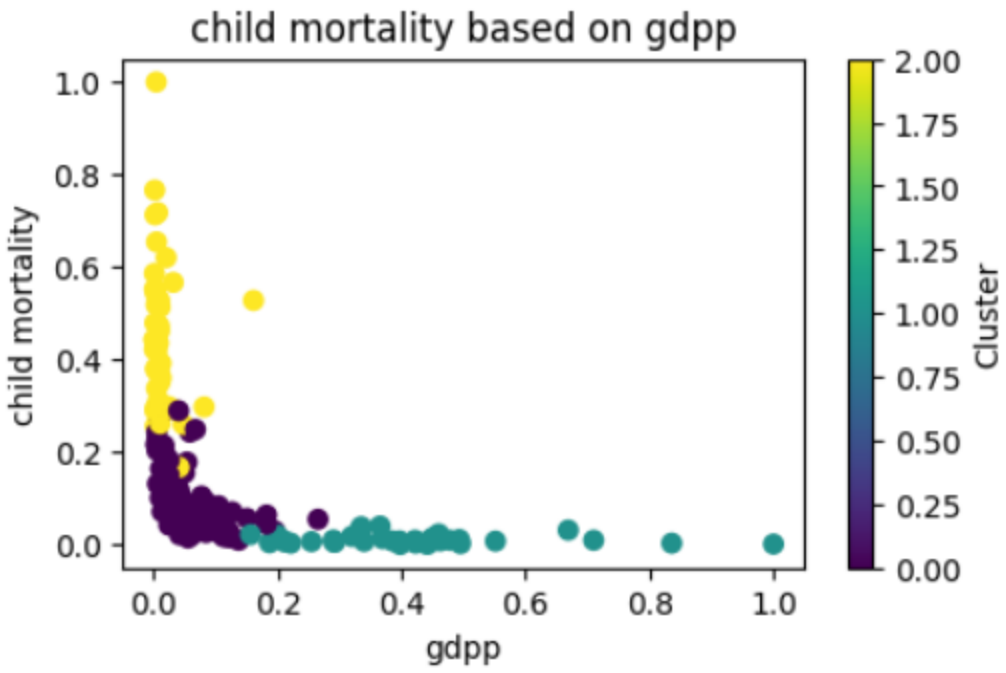
```
labels_pred = K_means_model.fit_predict(data)
silhouette = silhouette_score(data, labels_pred)
print(silhouette)
```

```
0.34265474105126204
```

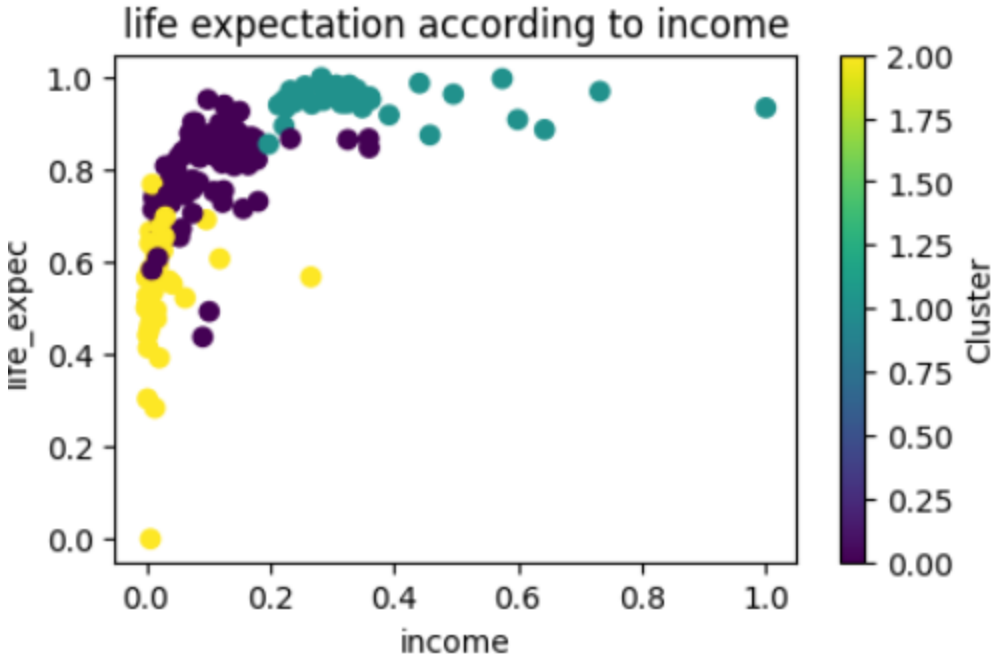
- کشور های توسعه یافته:خوشه ۱ ، کشورهای درحال توسعه با شرایط کمی بهتر: خوشه ۰ کشورهای به نسبت فقیر: خوشه ۲

قسمت 5

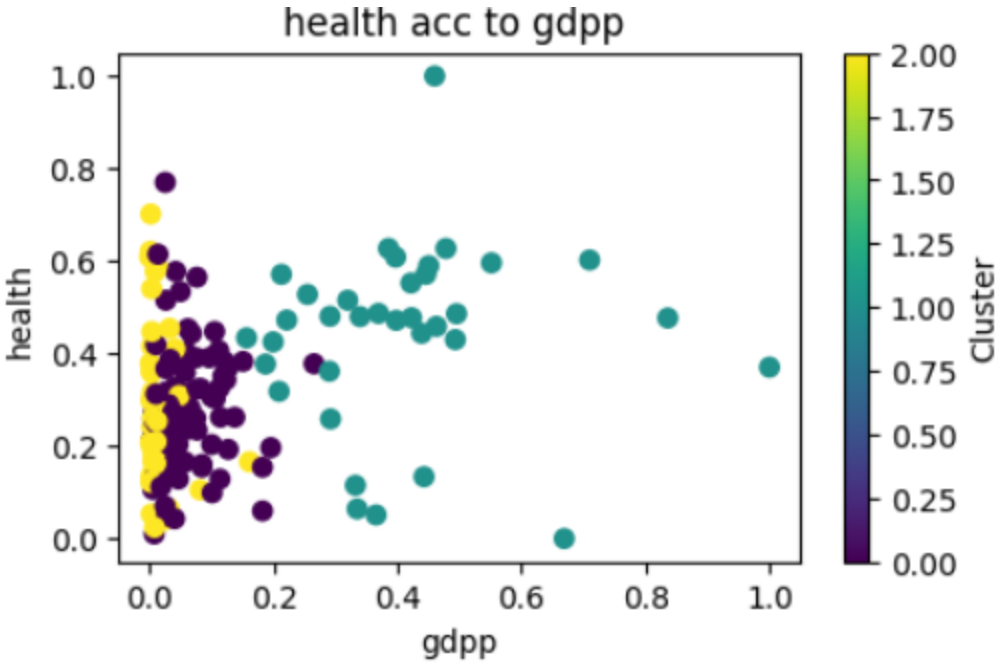
در این قسمت، سه ویژگی انتخاب شده و برای هر کدام نمودار **scatter** رسم شده است.



این نمودار رابطه بین مرگ و میر نوزادان و تولید ناخالص داخلی کشور ها را نشان می دهد. کشور های با **gdpp** بالا (خوشه سبز رنگ) **child_mortality** کمتری را داشته اند. اما کشور های خوشه زرد رنگ **gdpp** کمتر و مرگ و میر بالایی را داشته اند.



در نمودار روبرو رابطه یا همبستگی درآمد با امید به زندگی در کشور های مختلف با توجه به خوشه بندی انجام شده، نشان داده شده است. کشور های با خوشه سبز، درآمد بیشتر و امید به زندگی بیشتری داشته اند. اما کشور های با رنگ زرد درآمد کمتر و امید به زندگی متغیری را داشته اند که نشان دهنده اختلاف طبقاتی در آن کشور هاست.



نمودار روبرو رابطه بین وضعیت سلامت و تولید ناخالص داخلی در میان سه خوشه از کشور ها را نشان می دهد. کشور های با خوشه بنفش و زرد **gdpp** کمتری دارند اما وضعیت سلامت در این کشور ها متغیر بوده و با **gdpp** همبستگی کمی دارد(نشان دهنده جوامع طبقاتی)

اما کشور های خوشه سبز **gdpp** بالاتر و سلامت بهتری دارند و همبستگی **health** و **gdpp** در این خوشه بیشتر از خوشه های دیگر است.

قسمت PCA_1

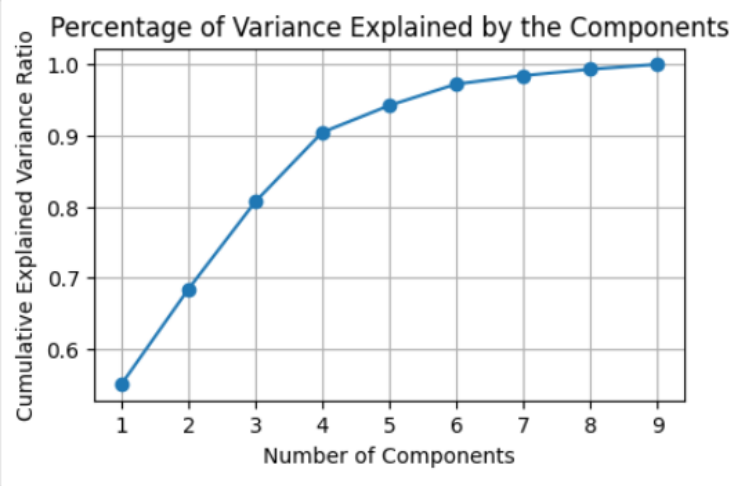
در این قسمت با استفاده از تحلیل مولفه های اصلی که یک روش کاهش بعد می باشد، مولفه های اصلی دیتاست شناسایی شده و میزان سهم آن ها از اطلاعات موجود در داده با استفاده از متد **_pca.explained_variance_ratio** نشان داه شده است.

```
[38]: print(pca.explained_variance_ratio_)

[0.55001227 0.13384784 0.12301053 0.09749047 0.03777964 0.03013659
 0.01190434 0.00887791 0.00694042]
```

قسمت PCA_2

در این قسمت، میزان پوشش یا توضیح تجمعی واریانس داده(**cumulative_variance_ratio**) بوسیله ی هر مولفه اصلی نشان داده شده است. این نمودار نشان می دهد با ۴ مولفه اصلی می توان ۹۰ درصد واریانس در دیتاست را پوشش داد. بنابراین ۴ مولفه اصلی به طور کافی توزیع داده ها را نشان می دهند.



قسمت PCA_3

در این قسمت یک دیتاست جدید از دیتاست قبلی که ابعاد آن با استفاده از روش **PCA** کاهش یافته ایجاد شده است که ویژگی های آن ۴ مولفه اولی هستند که در قسمت قبل در مورد آن توضیح داده شد.

قسمت PCA_4

در این قسمت، الگوریتم یک مدل **kmeans** بر روی دیتاست جدید ایجاد شده و عمل خوشه بندی روی آن انجام شده است. امتیاز **silhouette** این مدل با مدل قبلی مقایسه شده که نتیجه نشان میدهد الگوریتم روی دیتای کاهش‌بعدیافته عملکرد بهتری دارد. امتیاز مدل قبلی **0.34** بوده اما مدل جدی امتیاز **0.39** را دارد.

silhouette score

```
labels_pred = kmeans.fit_predict(data_reduced)
silhouette = silhouette_score(data_reduced, labels_pred)
print(silhouette)
```

```
0.39155410911751376
```

سوال ۲

قسمت 1

در این قسمت دیتاست خواسته شده با استفاده از متد **random.multivariate_normal** کتابخانه **numpy** تولید شده است که شامل دو دسته و دارای ۱۰ هزار قلم داده است. برای تولید مجموعه داده، از توزیع

گوسی (نرمال) با میانگین‌های مختلف و ماتریس کوواریانس یکسان استفاده می‌کنیم. داده‌ها را به دو کلاس تقسیم می‌کنیم به طوری که هر کلاس از یک توزیع گوسی با میانگین متفاوت و ماتریس کوواریانس یکسان پیروی می‌کند.

برای اطمینان از خطی جداسازی‌پذیر بودن داده‌ها، میانگین‌ها را به گونه‌ای انتخاب می‌کنیم که فاصله کافی بین دو کلاس وجود داشته باشد. به عنوان مثال:

میانگین کلاس اول **(2,2)**

میانگین کلاس دوم **(−2,−2)**

ماتریس کوواریانس **[[1,0],[0,1]]**

این تنظیمات باعث می‌شود که دو مجموعه داده به طور مناسب از هم جدا شوند و به راحتی قابل طبقه‌بندی باشند.

قسمت 2

در این قسمت شبکه های عصبی **perceptron** و **Adaline** ایجاد شده اند.

پرسپترون یک نوع ساده از شبکه‌های عصبی است که شامل یک لایه و یک تابع فعال‌سازی پله‌ای می‌شود. هدف پرسپترون طبقه‌بندی داده‌های ورودی به دو کلاس مختلف است.

ساختار:

۱. **ورودی‌ها** :ویژگی‌های ورودی .(x1,x2)(x_1, x_2)(x1,x2)
۲. **وزن‌های اولیه** :مقادیر کوچک تصادفی یا صفر.
۳. **تابع فعال‌سازی** :تابع فعال‌سازی پله‌ای که خروجی را بر اساس مقدار خطی ورودی تعیین می‌کند.

فرآیند:

- پرسپترون از یک تابع فعال‌سازی پله‌ای استفاده می‌کند که خروجی آن ۱ یا -۱ است.

- در هر تکرار(epoch) ، پرسپترون خطای طبقه‌بندی را محاسبه و بر اساس آن وزن‌ها و بایاس را به‌روزرسانی می‌کند.

آدالاین مشابه پرسپترون است اما از یک تابع فعال‌سازی خطی و تابع هزینه میانگین مربعات (MSE) استفاده می‌کند. هدف آدالاین نیز طبقه‌بندی داده‌ها به دو کلاس مختلف است.

ساختار:

- **ورودی‌ها** :ویژگی‌های ورودی .(x1,x2)(x_1, x_2)(x1,x2)
- **وزن‌های اولیه** :مقادیر کوچک تصادفی یا صفر.
- **تابع فعال‌سازی** :تابع فعال‌سازی خطی که خروجی آن مقدار پیوسته‌ای است.

فرآیند:

آدالاین از تابع فعال‌سازی خطی استفاده می‌کند که خروجی آن مقدار خطی ورودی است.در هر تکرار (epoch)، آدالاین خطای میانگین مربعات را محاسبه و با استفاده از روش گرادیان نزولی وزن‌ها و بایاس را به‌روزرسانی می‌کند.

تفاوت‌ها

تابع فعال‌سازی :پرسپترون از تابع پله‌ای استفاده می‌کند، در حالی که آدالاین از تابع خطی استفاده می‌کند.

به‌روزرسانی وزن‌ها :پرسپترون بر اساس خطای طبقه‌بندی وزن‌ها را به‌روزرسانی می‌کند، در حالی که آدالاین بر اساس کمینه‌سازی خطای میانگین مربعات وزن‌ها را به‌روزرسانی می‌کند.

این تفاوت‌ها باعث می‌شود که آدالاین به طور کلی پایداری بیشتری در فرآیند یادگیری داشته باشد و به طور نرم‌تری به خطای کمینه همگرا شود، در حالی که پرسپترون ممکن است به تغییرات کوچک در داده‌های ورودی حساس باشد.

قسمت 3

با استفاده از متد **train_test_split** دیتاست به مجموعه آموزشی و آزمایشی با اندازه ۸۰ به ۲۰ تقسیم شده است. بعد از آن هر کدام از مدل های **Perceptron** و **Adaline** آموزش دیده شده اند و روی داده تست عمل خوشه بندی آن ها انجام شده است و در نهایت دقت دو مدل مقایسه شده است.

accuracy

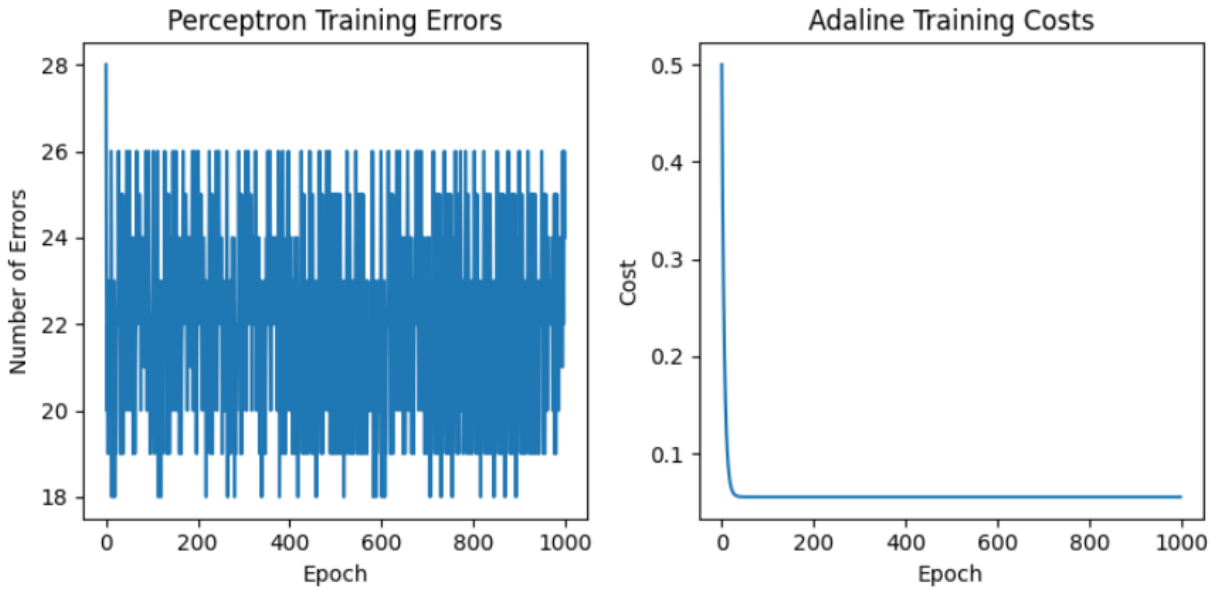
```
acc_perceptron = accuracy_score(y_test, y_pred_perceptron)
acc_adaline = accuracy_score(y_test, y_pred_adaline)

print(f"Perceptron Accuracy: {acc_perceptron}")
print(f"Adaline Accuracy: {acc_adaline}")
```

```
Perceptron Accuracy: 0.994
Adaline Accuracy: 0.9955
```

رسم نمودار خطاهای آموزشی

خطاهای آموزشی (پرسپترون) و هزینه‌ها (آدالاین) را در طول تکرارهای آموزشی رسم شده است.



نمودار فوق نشان میدهد دقت آدالاین معمولاً بالاتر است زیرا از روش کمینه‌سازی خطای میانگین مربعات استفاده می‌کند که به پایداری بیشتر مدل منجر می‌شود. آدالاین به دلیل استفاده از روش گرادیان نزولی، معمولاً به طور نرم‌تر و پایدارتر همگرا می‌شود.

برای داده‌های خطی جداسازی‌پذیر، هر دو مدل پرسپترون و آدالاین مناسب هستند، اما آدالاین به دلیل استفاده از تابع هزینه و روش گرادیان نزولی، معمولاً عملکرد بهتری دارد.

سوال ۳

قسمت a

در این تمرین که مربوط به یادگیری تقویتی است، قرار است عامل یادگیرنده **Agent** سیاست بهینه را برای تعامل با محیط و انتخاب تصمیم یا عمل مناسب، اتخاذ کند. یک سیاست **Policy** مجموعه ای از عمل ها **Action** در تعامل با محیط است. برای آموزش این عامل یادگیرنده از یک محیط شبیه سازی استفاده شده است. عامل بعد از یادگیری سیاست بهینه، با انجام عمل های آن سیاست، به سمت هدف بهینه حرکت میکند.

ابتدا کتابخانه‌های **Gym** و **NumPy** برای استفاده در محیط و انجام محاسبات ریاضی فراخوانی شده‌اند. با استفاده از کتابخانه **Gym**، محیط **Frozen Lake** ساخته و بازنشانی (**reset**) شده است تا آماده استفاده باشد.

تعدادی پارامتر برای الگوریتم تکرار مقداری تنظیم شده‌اند:

Num_iterations: تعداد تکرارهای الگوریتم

threshold مقدار آستانه برای بررسی همگرایی تابع مقدار

gamma: عامل تخفیف **(discount factor)**

برای هر حالتی که عامل یادگیرنده در آن قرار می گیرد، یک امتیاز عددی به آن حالت و عمل های آن حالت در یک جدولی به نام **Q-table** اختصاص داده می شود که این عمل با استفاده از تابعی تحت عنوان **Q-function** انجام می شود. برای هر حالت **(state)** محاسبات **Q-value** انجام می‌شود و بیشترین مقدار

انتخاب می‌شود. الگوریتم تا زمانی که تغییرات جدول مقداری کمتر از مقدار آستانه باشد، ادامه پیدا می‌کند. بعد از این مرحله، قرار است سیاست بهینه با استفاده از مقادیر جدول، استخراج شود.

ابتدا لازم مقادیر جدول در هر حالت و به ازای هر عمل پیدا شوند و با استفاده از آنها سیاست بهینه پیدا شود . بنابراین یک تابع به نام **value_iteration** نوشته شده که هدف اصلی این تابع، پیدا کردن یک جدول مقداری بهینه است که نشان می‌دهد هر حالت (**state**) در محیط چقدر ارزش دارد. این ارزش‌ها بر اساس پاداش‌های مورد انتظار در آینده محاسبه می‌شوند و به ما کمک می‌کنند تا بهترین سیاست را برای تصمیم‌گیری در هر حالت انتخاب کنیم.

تابع **value_iteration** در نهایت یک جدول مقداری بهینه تولید می‌کند که نشان می‌دهد ارزش هر حالت چقدر است. این جدول به ما کمک می‌کند تا بفهمیم در هر حالت کدام عمل یا اقدام بهترین انتخاب است تا به بیشترین پاداش ممکن دست پیدا کنیم. بعد از به دست آوردن این جدول، می‌توانیم از تابع **extract_policy** برای استخراج سیاست بهینه استفاده کنیم که مشخص می‌کند در هر حالت چه اقدامی باید انجام دهیم.

```python
def value_iteration(env):
    num_iterations = 1000
    threshold = 1e-20
    gamma = 1.0

    value_table = np.zeros(env.observation_space.n)

    for i in range(num_iterations):

        updated_value_table = np.copy(value_table)

        for s in range(env.observation_space.n):

            Q_values = [sum([prob*(r + gamma * updated_value_table[s_])
                             for prob, s_, r, _ in env.P[s][a]])
                                   for a in range(env.action_space.n)]

            value_table[s] = max(Q_values)

        if (np.sum(np.fabs(updated_value_table - value_table)) <= threshold):
             break

    return value_table
```

یک تابع به نام **extract_policy** برای این کار نوشته شده است. در این تابع، برای هر حالت، **Q-value**های مربوط به هر عمل (**action**) محاسبه می‌شود و عمل با بیشترین **Q-value** به‌عنوان سیاست بهینه انتخاب می‌شود.

```python
def extract_policy(value_table):

    gamma = 1.0
    policy = np.zeros(env.observation_space.n)

    for s in range(env.observation_space.n):
        Q_values = [sum([prob*(r + gamma * value_table[s_])
                                for prob, s_, r, _ in env.P[s][a]])
                                    for a in range(env.action_space.n)]

        policy[s] = np.argmax(np.array(Q_values))

    return policy
```

این تابع به ما کمک می‌کند تا بهترین سیاست را برای حرکت در محیط **Frozen Lake** پیدا کنیم و به‌طور موثر به هدف برسیم.

درنهایت پس از اجرای توابع فوق، سیاست بهینه برای این عامل استخراج شده است.

```python
optimal_policy = extract_policy(optimal_value_function)
print(optimal_policy)
```

```
[0. 3. 3. 3. 0. 0. 0. 0. 3. 1. 0. 0. 0. 2. 1. 0.]
```

عمل **0** حرکت به سمت چپ

عمل **1** حرکت به سمت پایین

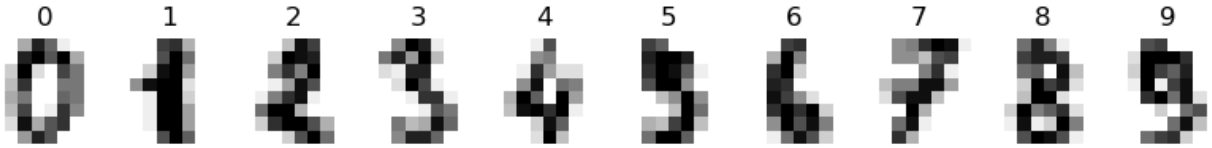
عمل **2** حرکت به سمت بالا

عمل **3** حرکت به سمت راست

سوال ۴

قسمت a

در مرحله اول کتابخانه های لازم برای کار با این دیتاست فراخوانی می‌شوند. بعد از لود کردن دیتاست، برای هر کدام از تارگت ها نمودار آن ها رسم شده است.



قسمت b

در این قسمت، مراحل پیش پردازش داده ها انجام شده است. نرمالسازی و استانداردسازی مقیاس عددی داده ها به عنوان این مراحل با استفاده از ماژول **StandardScaler** انجام شده است. با انجام این مراحل داده ها در یک مقیاس استاندارد قرار می گیرند که کار مدل یادگیرنده برای کار با این داده ها راحت تر می شود.

قسمت c

در این قسمت یک مدل خوشه بندی روی دیتاست ایجاد شده است که از الگوریتم **K-Means** برای خوشه‌بندی داده‌ها استفاده می‌شود و سپس خوشه‌های به دست آمده به برچسب‌های اصلی داده‌ها نگاشت می‌شوند.

کتابخانه **KMeans** از **scikit-learn** برای اجرای الگوریتم **K-Means** فراخوانی شده است.

یک شیء از کلاس `KMeans` ایجاد می‌شود و پارامترهای زیر به آن داده می‌شود:

n_clusters=num_clusters: تعداد خوشه‌ها (در اینجا ۱۰)

n_init=10: تعداد تکرار الگوریتم با مجموعه‌های مختلفی از مراکز ابتدایی

random_state=42: برای تثبیت نتایج و قابل تکرار بودن

مدل بر روی داده‌های ورودی **data** آموزش داده می‌شود و خوشه‌های پیش‌بینی شده را برمی‌گرداند. نتیجه این تابع یک آرایه است که نشان می‌دهد هر نمونه به کدام خوشه تعلق دارد

قسمت d

در این بخش از کد، الگوریتم خوشه‌بندی سلسله‌مراتبی (**Hierarchical Clustering**) با استفاده از کتابخانه‌های **scikit-learn** و **scipy** پیاده‌سازی شده است. این کد شامل مراحل خوشه‌بندی داده‌ها و سپس نگاشت خوشه‌ها به برچسب‌های اصلی است. ماژول **AgglomerativeClustering** از کتابخانه **scikit-learn** برای اجرای الگوریتم خوشه‌بندی سلسله‌مراتبی استفاده می‌شود.

در پایان، **hierarchical_predicted_labels** آرایه‌ای است که برچسب‌های پیش‌بینی شده برای داده‌های ورودی را شامل می‌شود و می‌توان آن را با برچسب‌های اصلی برای ارزیابی دقت مدل مقایسه کرد.

قسمت e

در این قسمت ارزیابی مدل ها با استفاده از شاخص **Accuracy** نشان داده شده است که نشان می دهد عملکرد مدل **hierarchical** بهتر بوده است.

```
]: accuracy = accuracy_score(digits.target, predicted_labels)
   print(accuracy)
```

```
0.6210350584307178
```

```
]: hierarchical_accuracy = accuracy_score(digits.target, hierarchical_predicted_labels)
   print(hierarchical_accuracy)
```

```
0.7579298831385642
```

قسمت f

در این تابع **map_clusters_to_labels**، با استفاده از مجموعه‌ای از خوشه‌ها (**clusters**) و برچسب‌های واقعی (**true_labels**)، ما یک نگاشت از خوشه‌ها به برچسب‌های واقعی را ایجاد می‌کنیم.

هر خوشه با استفاده از برچسب اکثریتی که در آن خوشه وجود دارد، به یک برچسب واقعی نگاشت می‌شود. به عبارت دیگر، ما برچسب‌های واقعی متناظر با هر خوشه را محاسبه و آن‌ها را برمی‌گردانیم.

این تابع می‌تواند برای اختصاص برچسب‌های پیش‌بینی شده به هر خوشه در یک مسأله خوشه‌بندی استفاده شود.

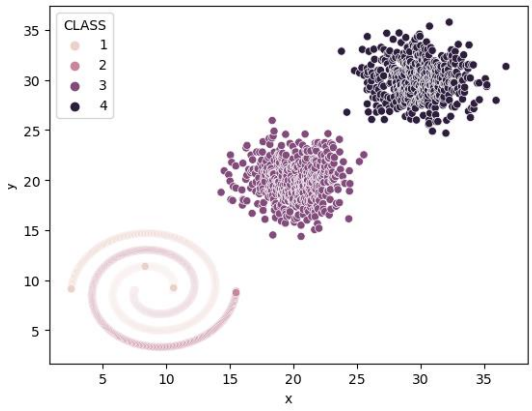
```
def map_clusters_to_labels(clusters, true_labels):
    label_map = np.zeros_like(clusters)
    for i in range(num_clusters):
        mask = (clusters == i)
        most_common = np.bincount(true_labels[mask]).argmax()
        label_map[mask] = most_common
    return label_map

predicted_labels = map_clusters_to_labels(clusters, digits.target)
```

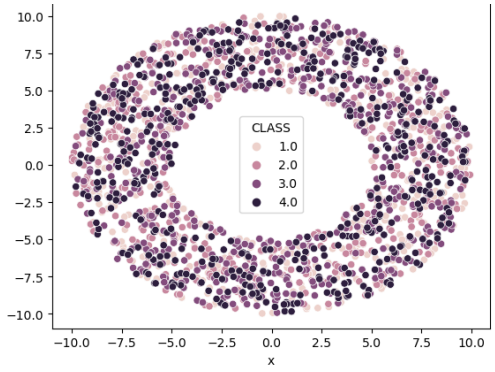
سوال ۵

قسمت a

در این قسمت، دیتاست ها لود شده و برای هر کدام از آن ها نمودار رسم شده است. نمودار اول نشان می دهد که دیتاست مورد نظر را در ۳ دسته خوشه بندی کرد .



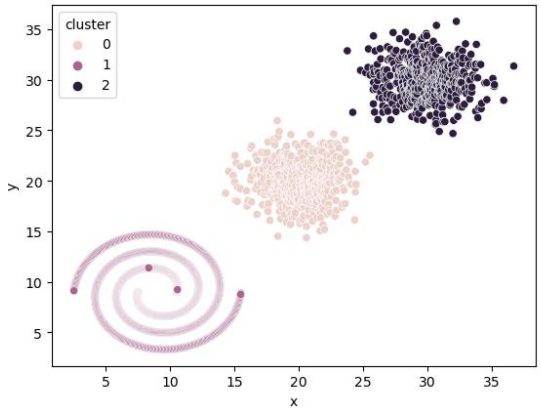
برای نمودار دوم به صورت شهودی عددی به عنوان تعداد خوشه بندی های احتمالی نمی توان یافت.

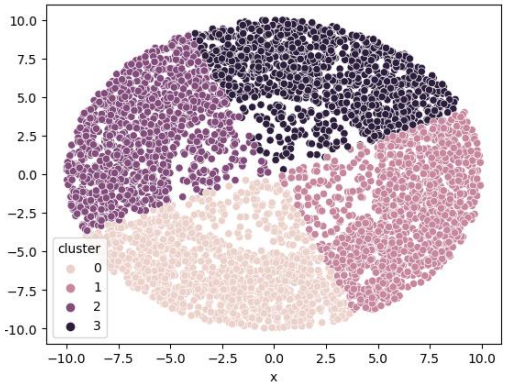


قسمت b

در این قسمت یک دو مدل خوشه بندی روی دیتاست ها ایجاد شده است که از الگوریتم **K-Means** برای خوشه‌بندی داده‌ها استفاده می‌شود و سپس خوشه‌های به دست آمده به برچسب‌های اصلی داده‌ها نگاشت می‌شوند.

کتابخانه **KMeans** از **scikit-learn** برای اجرای الگوریتم **K-Means** فراخوانی شده است. نمودار هر یک از دیتاست ها بعد از خوشه بندی نشان داده شده است.





نمودار روبه رو نشان می دهد دیتاست دوم را می توان به ۴ خوشه تقسیم کرد.

قسمت C

کیفیت خوشه بندی با استفاده از یک معیار به نام **silhouette_score** اندازه گیری می شود. که برای هر دو مدل خوشه بندی نشان داده شده است.

```
labels1 = data['cluster']
silhouette_score1 = silhouette_score(data, labels1)
print(silhouette_score1)
```

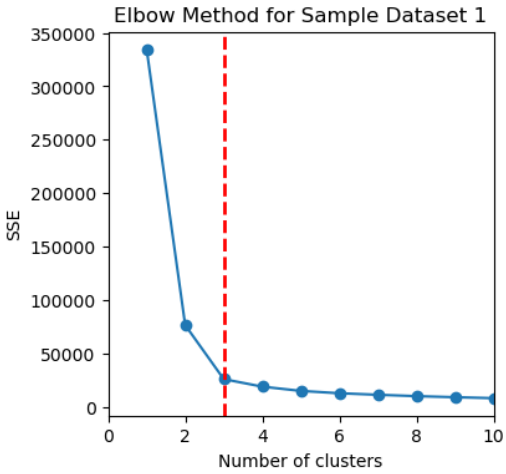
```
0.6972698629951405
```

```
labels2 = data2['cluster']
silhouette_score2 = silhouette_score(data2, labels2)
print(silhouette_score2)
```

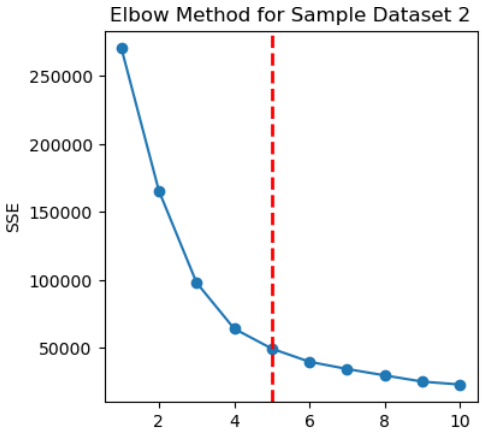
```
0.4227156774248591
```

این معیار نشان می دهد که کیفیت خوشه بندی روی دیتاست اول بهتر بوده است.

در ادامه بهینه ترین تعداد خوشه ها با استفاده از روش **Elbow** نشان داده شده است. در این روش در هر گام یک مدل **Kmeans** خوشه بندی بر اساس تعداد خوشه های مختلف ایجاد شده مقدار **SSE** برای آن ها را محاسبه می کند این روش بر اساس مقدار **"درون گروهی"** (**Sum of Squared Errors**، **SSE**) برای هر تعداد خوشه محاسبه می‌شود و تغییرات آن نسبت به تعداد خوشه‌ها را نشان می‌دهد. انجام این روش برای دیتاست اول نشان می دهد بهترین مقدار برای تعداد خوشه ها می تواند ۳ باشد که با خط عمودی قرمز نشان داده شده است.



همچنین این عمل برای دیتاست دوم انجام شده است که نمودار آن نشان می دهد بهترین مقدار می تواند عدد ۵ باشد.



قسمت d

در این قسمت از روش **DBSCAN** استفاده شده است. روش **DBSCAN** یک الگوریتم خوشه‌بندی است که بر پایه چگالی داده‌ها عمل می‌کند. این الگوریتم می‌تواند خوشه‌هایی با اشکال و اندازه‌های متفاوت را تشخیص دهد و همچنین به طور موثر با داده‌های پرت (نویز) مقابله کند. مهمترین ویژگی این الگوریتم این است که تعداد خوشه‌ها را مستقیماً مشخص نمی‌کند و به‌جای آن بر اساس چگالی داده‌ها خوشه‌ها را شناسایی می‌کند.

پارامترهای مهم:

Eps و min_samples

برای ارزیابی از روش **DBSCAN** بر روی داده‌ها استفاده می‌شود. این تابع به طول دو عدد اندازه میدهد که یک دیتاست و مقادیر مختلف برای پارامترهای **eps** و **min_samples** هستند. سپس از ترکیب مقادیر مختلف این پارامترها برای **DBSCAN** استفاده می‌کند و بر اساس ارزیابی‌هایی مانند امتیاز **silhouette** بهترین پارامترها را انتخاب می‌کند.

مدل **DBSCAN** بر روی داده ها آموزش دیده شده است و بهترین مقادیر برای پارامتر های آن بدست آمده است.

```
best_params, best_score, labels1 = evaluate_dbscan(data, eps_values, min_samples_values)
```

```
print(best_params)
print(best_score)
```

```
{'eps': 1.4000000000000001, 'min_samples': 2}
0.3698406296462119
```

```
best_params, best_score, labels2 = evaluate_dbscan(data2, eps_values, min_samples_values)
```

```
print(best_params)
print(best_score)
```

```
{'eps': 1.1, 'min_samples': 2}
0.3091988003463327
```

این مقادیر پارامترها به عنوان بهترین پارامترها برای الگوریتم **DBSCAN** انتخاب شده‌اند.

دیتاست اول:

Eps: 1.4

این پارامتر نشان دهنده شعاع یا محدوده همسایگی است که برای تشخیص همسایگی نقاط استفاده می‌شود. در این حالت، فاصله بین هر نقطه و نقاط همسایه آن تا حداکثر ۱.۱ واحد است.

Min_samples: 2

این پارامتر نشان دهنده تعداد حداقل نقاطی است که برای تشکیل یک خوشه لازم است.

در این حالت، حداقل برای تشکیل یک خوشه نیاز است که حداقل ۲ نقطه در محدوده **eps** قرار داشته باشند.

دیتاست دوم:

Eps: 1.1

Min_samples: 2

قسمت e

در این قسمت ارزیابی مدل **DBSCAN** قسمت قبل با استفاده از معیار های **silhouette** و **davies_bouldin_score** انجام شده است.

dataset1

```
silhouette_avg = silhouette_score(data, labels1)

db_index = davies_bouldin_score(data, labels1)

print(f"Silhouette Score: {silhouette_avg}")
print(f"Davies-Bouldin Index: {db_index}")
```

```
Silhouette Score: 0.36212596464702557
Davies-Bouldin Index: 2.6731234991594324
```

dataset2 ¶

```
silhouette_avg = silhouette_score(data2, labels2)

db_index = davies_bouldin_score(data2, labels2)

print(f"Silhouette Score: {silhouette_avg}")
print(f"Davies-Bouldin Index: {db_index}")
```

```
Silhouette Score: 0.24304971911192094
Davies-Bouldin Index: 1.1294843258105451
```

معیار های ارزیابی نشان می دهد عملکرد مدل **DBSCAN** بر روی دیتاست اول بهتر بوده است.

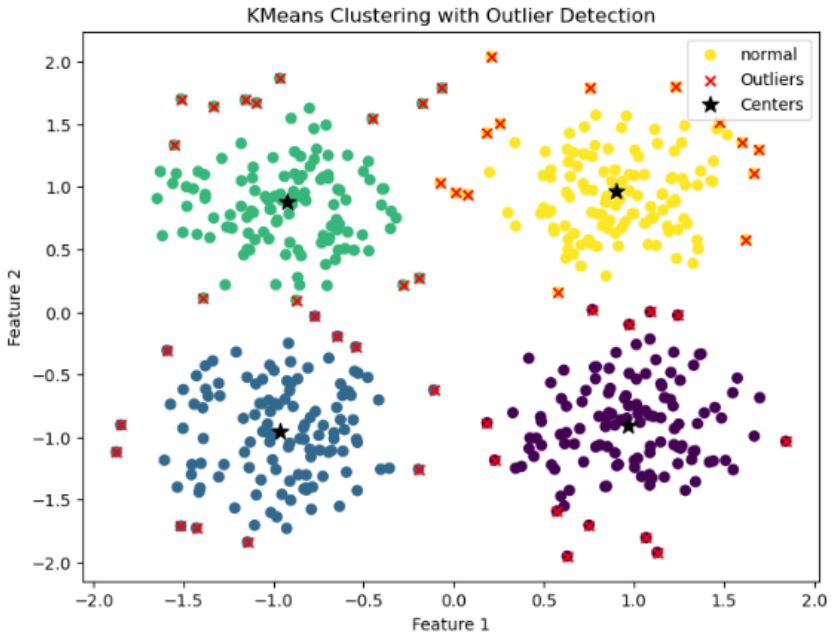
قسمت f

شناسایی داده های پرت با استفاده از **Kmeans** امکان پذیر است. نقاط داده ای که از مراکز خوشه ها دور هستند، به عنوان داده های پرت شناسایی می شوند.

ابتدا مراکز خوشه‌ها را با استفاده از الگوریتم **KMeans** محاسبه می‌کند. سپس، برای هر نقطه داده‌ای، فاصله آن را تا مرکز خوشه نزدیکترین محاسبه می‌کند. اگر فاصله یک نقطه داده‌ای از مرکز خوشه بسیار بزرگتر از میانگین فواصل تمامی نقاط باشد، این نقطه به عنوان یک داده پرت شناخته می‌شود.

بنابراین، داده‌هایی که فاصله آن‌ها از مرکز خوشه بسیار بیشتر از معمول است و به طور کلی از سایر نقاط داده‌ای در خوشه‌ها دورتر قرار دارند، به عنوان داده‌های پرت شناسایی می‌شوند. این داده‌ها اغلب به عنوان نقاطی که از الگوریتم نمونه‌گیری به طور اشتباه شناسایی شده‌اند یا دارای ویژگی‌های غیرمعمول هستند توصیف می‌شوند.

برای انجام این روش، ابتدا دیتاست سوم لود شده عمل پیش پردازش روی آن انجام شده و نمودار آن رسم شده است. سپس مدل **kmeans** روی دیتاست ایجاد شده است. در این مدل، فواصل هر نقطه از مرکز خوشه‌ای که به آن نقطه اختصاص یافته است محاسبه می‌شود. نقاطی که فاصله آن‌ها از مرکز خوشه‌ها بیشتر از آستانه تعیین شده است، به عنوان پرت تشخیص داده می‌شوند. در نهایت نمودار دیتاست بعد از اعمال مدل **Kmeans** رسم شده است که در آن نقاط پرت نشان داده شده است.



سوال ۶

ابتدا تصویر با استفاده از کتابخانه **Image** بارگذاری شده است. بعد از آن تصویر به یک بردار عددی تبدییل شده است. یک مدل خوشه بندی **Kmeans** با پارامتر های مختلف ایجاد خواهد شد.

خوشه‌بندی **K-means** را با مقادیر مختلف **K** (۱، ۲، ۴، ۸، ۱۶، ۳۲) اعمال می‌کنیم. این الگوریتم پیکسل‌های تصویر را به **K** دسته تقسیم می‌کند و هر دسته یک مقدار میانگین (**Centroid**) خواهد داشت.

از مقادیر میانگین خوشه‌ها برای بازسازی تصویر استفاده می‌کنیم. هر پیکسل تصویر با نزدیکترین مقدار میانگین جایگزین می‌شود.

نتیجه اعمال خوشه بندی با پارامتر های **k** مختلف به صورت زیر است:

از تصاویر فوق مشخص است که هر چقدر تعداد خوشه ها بیشتر باشد، تصاویر به تصویر اصلی شبیه تر هستند و اندازه آن ها بیشتر خواهد بود.