خوشه بندی کاراکتر ها با استفاده از الگوریتم SOM

الگوریتم **SOM**:

- الگوریتم *(Self-Organizing Map)* یکی از معروفترین مدل های شبکه ی عصبی است به شبکه ی الگوریتم های رقابتی تعلق دارد.
الگوریتم SOM مبتنی بر یادگیری *unsupervised* و رقابتی است.
این الگوریتم به این صورت عمل می کند که در ابتدا مختصات داده ها را میگیرد و حالا بسته به تعداد دسته بندی ها نقاط رندوم ایجاد می کند.

- مرحله ی اول:
در این مرحله هر داده نزدیکترین نقطه از نقاط رندوم که به اصطلاح وزن های ما(یا مراکز خوشه های ما) هستند را پیدا می کند و در خوشه ی این نقطه قرار می گیرد

- مرحله دوم:
در این مرحله هر کدام از این نقاط رندوم بر اساس داده ای که به آن پیوست شده مقدار خود را با فرمول زیر آپدیت می کند:

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha[x_i - w_{ij}(\text{old})].$$

که در این فرمول w نشانگر مرکز دسته است و X داده ای است که w نزدیکترین مرکز دسته به آن محسوب شده است و آلفا نرخ یادگیری است که به مرور زمان و افزایش اپیک ها , کوچکتر می شود. و می توانیم این دو مرحله را برای تمام داده ها انقدر اجرا کنیم که به یک مقدار خطای ثابت برسیم ولی چون الگوریتم unsupervised است پس گذاشتن شرط خطا خیلی منطقی نیست و من در پیاده سازی که انجام دادم این الگوریتم را برای مقدار های مشخص اجرا کردم.

در ابتدا تمامی ورودی ها در یک لیست *sample* ذخیره می کنیم.

و یک لیست *clusters* و یک آرایه *minimum* خارج از توابع تعریف می کنیم.

من این تمرین را در یک تابع به صورت زیر تعریف کردم:

در ابتدا ۲۵ نقطه ی رندوم در ابعاد داده ها تولید می کنیم و به لیست *clusters* اضافه می کنیم.

سپس الگوریتم زیر را برای نمونه ۱۰۰۰ بار اجرا می کنیم:

در ابتدا مقدار آلفا یا ضریب یادگیری را برابر با ۰.۶ قرار دادم و هر بار با اجرای کامل الگوریتم این مقدار نصف می شود (چون هر چقدر پیش می رویم سرعت همگرایی کمتر می شود ضریب یادگیری بالا ممکن است باعث شود هیچ وقت الگوریتم ما همگرا نشود)

ارایه *distance* یک آرایه یا بردار ۲۵ تایی است که هر بار برابر است با فاصله ی اقلیدسی داده ی i از تمام مرکز خوشه ها.

بعد از این که به ازای هر داده فواصل اقلیدسی را از تمام کلاستر ها پیدا کردیم، مرکز خوشه ای که کمترین فاصله را داشته باشد به عنوان خوشه برنده اعلام می کنیم. و این خوشه با استفاده از روش آپدیت بالا آپدیت می شود.

```
def findmindis (clusters, sample, minimum):
    character = ["A1","B1","C1","D1","E1","J1","K1","A2","B2","C2","D2","E2","J2","K2","A3","B3","C3","D3","E3","J3","K3"]
    for k in range (25):
        cluster = np.random.rand(63,)
        clusters.append(cluster)
    Alpha = 0.6
    epoch = 0
    while epoch < 1000 :
        Alpha = Alpha*0.5
        distance = np.zeros((25,))
        subtract = np.zeros((63,), dtype=int)
        for i in range (21):
            for j in range (25):
                subtract =  np.subtract(sample[i], clusters[j])
                distance[j] = np.inner(subtract.transpose(),subtract)
            k = distance.argmin()
            minimum[i] = k
            subtract =  np.subtract(sample[i], clusters[k])
            clusters[k] = clusters[k] + Alpha*subtract
        epoch += 1
    for i in range (21):
        print ( str(character[i]) + " is belong to cluster " + str(minimum[i]) )
    return clusters
```

و همانطور که گفته شد این الگوریتم را برای تعداد دفعات مختلف اجرا می کنیم تا نتیجه ها را بتوانیم مقایسه کنیم.

epoch = 1000

```
>>> main()
A1 is belong to cluster 17.0
B1 is belong to cluster 10.0
C1 is belong to cluster 5.0
D1 is belong to cluster 10.0
E1 is belong to cluster 10.0
J1 is belong to cluster 12.0
K1 is belong to cluster 10.0
A2 is belong to cluster 17.0
B2 is belong to cluster 5.0
C2 is belong to cluster 5.0
D2 is belong to cluster 5.0
E2 is belong to cluster 5.0
J2 is belong to cluster 12.0
K2 is belong to cluster 5.0
A3 is belong to cluster 23.0
B3 is belong to cluster 10.0
C3 is belong to cluster 5.0
D3 is belong to cluster 10.0
E3 is belong to cluster 10.0
J3 is belong to cluster 12.0
K3 is belong to cluster 10.0
```

epoch = 10000

```
>>> main()
A1 is belong to cluster 5.0
B1 is belong to cluster 11.0
C1 is belong to cluster 11.0
D1 is belong to cluster 11.0
E1 is belong to cluster 11.0
J1 is belong to cluster 23.0
K1 is belong to cluster 5.0
A2 is belong to cluster 5.0
B2 is belong to cluster 11.0
C2 is belong to cluster 11.0
D2 is belong to cluster 11.0
E2 is belong to cluster 11.0
J2 is belong to cluster 23.0
K2 is belong to cluster 16.0
A3 is belong to cluster 5.0
B3 is belong to cluster 11.0
C3 is belong to cluster 11.0
D3 is belong to cluster 23.0
E3 is belong to cluster 11.0
J3 is belong to cluster 23.0
K3 is belong to cluster 5.0
>>>
```

آرایش خطی و لوزی:

خطی: در آرایش خطی در اجرای الگوریتم بالا اگر به عنوان مثال خوشه i انتخاب شد خوشه i+1 و خوشه i-1 هم (در صورت وجود با توجه به الگوریتم به روز رسانی ، آپدیت می شوند)

```
if k > 0 :
    subtract1 =  np.subtract(sample[i], clusters[k-1])
    clusters[k-1] = clusters[k-1] + Alpha*subtract1
if k < 24:
    subtract2 =  np.subtract(sample[i], clusters[k+1])
    clusters[k+1] = clusters[k+1] + Alpha*subtract2
```

نمونه اجرای آرایش خطی را برای ۱۰۰۰ بار اجرای الگوریتم در زیر مشاهده می کنید:

```
A1 is belong to cluster 16.0
B1 is belong to cluster 2.0
C1 is belong to cluster 0.0
D1 is belong to cluster 3.0
E1 is belong to cluster 3.0
J1 is belong to cluster 18.0
K1 is belong to cluster 5.0
A2 is belong to cluster 16.0
B2 is belong to cluster 1.0
C2 is belong to cluster 0.0
D2 is belong to cluster 0.0
E2 is belong to cluster 1.0
J2 is belong to cluster 18.0
K2 is belong to cluster 14.0
A3 is belong to cluster 16.0
B3 is belong to cluster 3.0
C3 is belong to cluster 0.0
D3 is belong to cluster 5.0
E3 is belong to cluster 3.0
J3 is belong to cluster 18.0
K3 is belong to cluster 5.0
```

لوزی: در این مدل تمام کلاستر ها را در یک ماتریس مربعی قرار می دهیم و در مرحله ای که به عنوان مثال *cluster(i,j)* برنده اعلام شود ، کلاسترهای *(i,j+1) ((i-1,j+1) (i+1,j) (i,j-1)* هم آپدیت می شوند.

من در پیاده سازی این نوع آرایش اینگونه عمل کردم که چون تعداد مرکز خوشه ها ۲۵ تا بود ، ماتریس ایجاد شده یک ماتریس ۵*۵ می شود پس به عنوان مثال اگر *cluster(i)* انتخاب بشود:

خوشه i+1 : در صورتی که i مضرب ۵ نباشد انتخاب می شود

خوشه ی i-1 : در صورتی که i به پیمانه ی ۵ برابر ۱ نباشد

خوشه ی i+5 : اگر i <= 20

خوشه ی i-5 : اگر i >= 5

با شرایط بالا این ۴ خوشه هم آپدیت می شوند.

نمونه اجرای الگوریتم برای ۱۰۰۰ دور در پایین دیده می شود.

```
>>> main()
A1 is belong to cluster 2.0
B1 is belong to cluster 16.0
C1 is belong to cluster 4.0
D1 is belong to cluster 10.0
E1 is belong to cluster 16.0
J1 is belong to cluster 0.0
K1 is belong to cluster 6.0
A2 is belong to cluster 2.0
B2 is belong to cluster 8.0
C2 is belong to cluster 4.0
D2 is belong to cluster 8.0
E2 is belong to cluster 8.0
J2 is belong to cluster 0.0
K2 is belong to cluster 14.0
A3 is belong to cluster 2.0
B3 is belong to cluster 12.0
C3 is belong to cluster 4.0
D3 is belong to cluster 12.0
E3 is belong to cluster 16.0
J3 is belong to cluster 0.0
K3 is belong to cluster 6.0
>>>
```

"همانطور که دیده می شود تعداد اپک ها وقتی از یک حدی بیشتر شود دقت دسته بندی ها خیلی تغییر نمی کند و همچنین تغییر آرایش آپدیت نیز تاثیر خاصی در این مثال ندارد. در دسته بندی کاراکتر ها بیشتر مقدار رندوم اولیه تاثیر گذار هست به همین خاطر مثلا اگر از هر کاراکتر یکی را به عنوان مرکز خوشه انتخاب کنیم تا این روال انتخاب رندوم مرکز خوشه ها را هدف دار تر کنیم احتمالا نتیجه ی بهتری خواهیم گرفت."