به نام خدا

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر

پروژه نهایی درس یادگیری ماشین

استاد:

دکتر احسان ناظرفرد

دانشجو:

حلیمه رحیمی

شماره دانشجویی:

۹۹۱۳۱۰۴۳

زمستان ۱۳۹۹

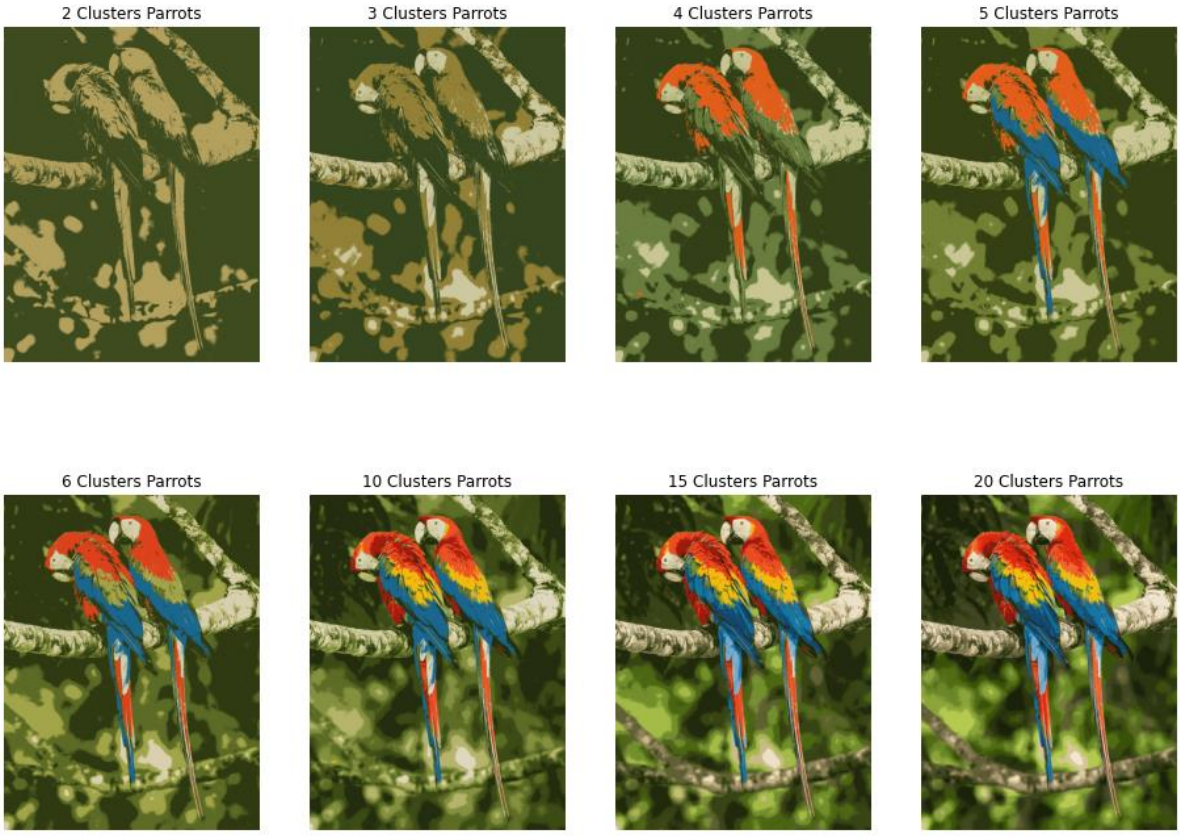
۱- با استفاده از یک کتابخانه‌ی آماده که در آن الگوریتم خوشه‌بندی K-Means وجود دارد، موارد زیر را پیاده‌سازی نمایید.

۱,۱) تصاویر bee.jpg و parrots.jpg را خوانده و نمایش دهید. هر تصویر از تعدادی پیکسل ساخته شده است و رنگ هر پیکسل با استفاده از ترکیب سه رنگ قرمز، سبز و آبی (RGB) ساخته می‌شود؛ به همین دلیل بعد از خواندن تصویر، مشاهد می‌کنید که تصویر خوانده شده یک ماتریس با مشخصات W*H*3 است به طوری که W و H اشاره به عرض و طول تصویر دارد و ۳ نشان دهنده‌ی هر کدام از سه رنگ RGB است. بنابراین، پیکسل‌های تصویر، داده‌های مورد نیاز مسئله می‌باشند که هر کدام دارای سه ویژگی هستند. پیکسل‌ها را با تعداد خوشه‌های ۲,۳,۴,۵,۶,۱۰,۱۵,۲۰ خوشه‌بندی کنید.

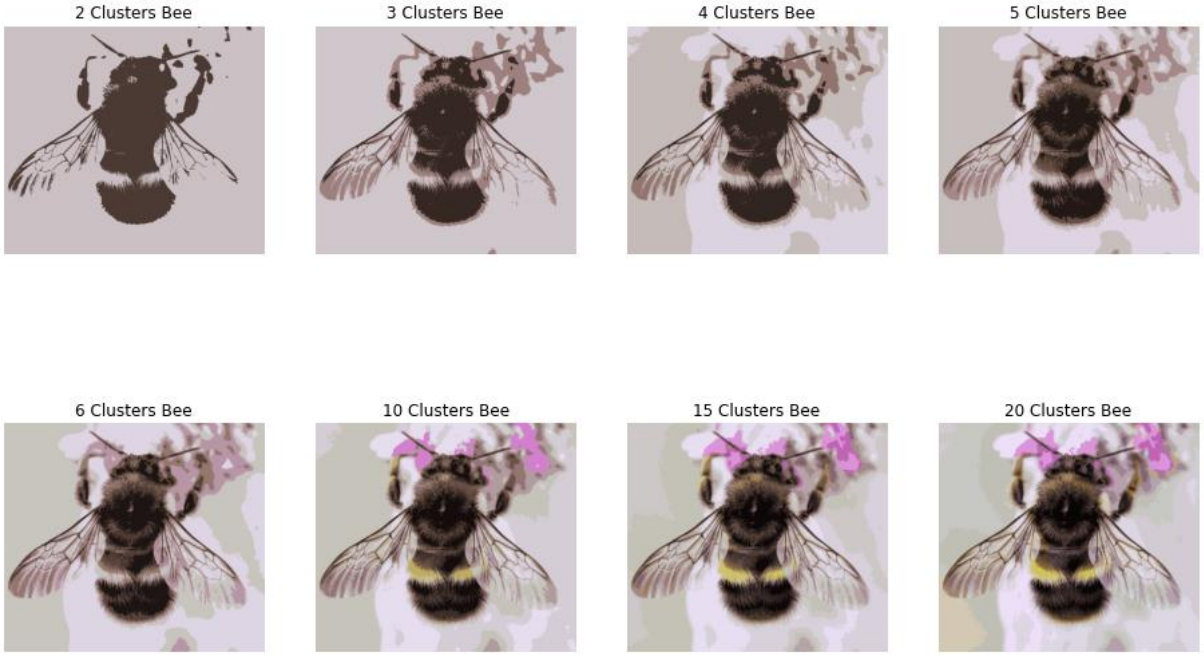
بعد از هر بار خوشه‌بندی تصاویر، رنگ پیکسل‌ها را با رنگ مرکز خوشه‌ای که در آن قرار می‌گیرند جایگزین کنید و تصویر حاصل را نمایش دهید.

برای این کار از کتابخانه‌ی sklearn و از k-means++ استفاده کردم. همچنین برای راحتی کار در ابتدا تابعی برای خوشه بندی و رنگ کردن تعریف کردم سپس با استفاده از حلقه به ازای k های متخلف آن را به اجرا درآوردم.

نتایج را در زیر مشاهده می‌کنید.



همانطور که مشهود است، با دو خوشه، پیکسل‌های با رنگ نزدیک به قرمز و زرد و سفید در یک خوشه و آنها که نزدیک به رنگ سبز و آبی و مشکی بوده اند در خوشه‌ی دیگر قرار گرفته اند. با بیشتر شدن تعداد خوشه‌ها در ابتدا بخش های روشن‌تر و نزدیک به سفید جدا شده اند و در k=۵ رنگ های قرمز، سبز و آبی، بعلاوه‌ی طیف تیره‌ای از سبز و رنگی مایل به سفید را داریم (سه رنگ اول همان RGB). در نهایت با ۲۰ خوشه تصویر به اصل خود نزدیک‌تر می‌شود.



در اینجا ابتدا به دلیل سیاهی بخشی از تصویر، رنگ سیاه اثر زیادی بر روی رنگ مرکز خوشه گذاشته است، در حالیکه در تصویر قبلی سبز بسیار تاثیرگذاری بوده و در ابتدا بخش تاریک تصاویر تقریبا سبز یشمی دیده می‌شد. با بیشتر شدن تعداد خوشه‌ها کم کم این تصویر به اصل خود نزدیک می‌شود.

۲,۱) در این بخش مجموعه داده‌ی Shill Bidding Dataset.csv را بارگذاری کنید.

الف) یکی از روش‌های تعیین تعداد خوشه‌های بهینه در الگوریتم K-Means استفاده از روش elbow است؛ این روش را توضیح دهید.

در خوشه بندی با بالا رفتن تعداد خوشه ها، (به علت خوشه‌های بیشتر) مدل هر بار بهتر بر روی داده‌ها برازش خواهد شد. تعداد بیشتر خوشه‌ها موجب می‌شود کم کم تعداد کمتری از داده‌ها در خوشه‌ها قرار بگیرند تا جایی که k=n هر داده را در یک خوشه قرار می‌دهد و واریانس درون خوشه در این حالت به صفر می‌رسد. این افزایش تعداد خوشه از جایی به بعد باعث بیش برازش می‌شود.

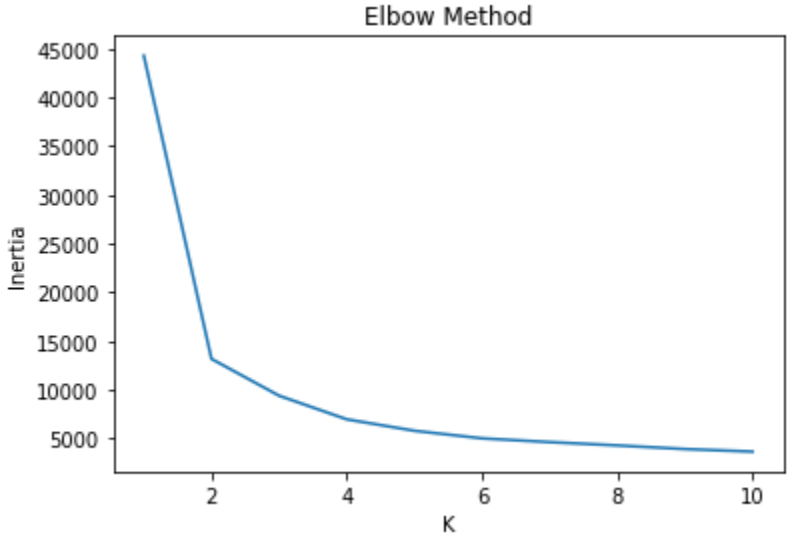
ایده‌ی روش elbow بر پایه‌ی این است که اولین تقسیم‌بندی‌های خوشه‌ها واریانس بیشتری را بین داده‌ها در نظر می‌گیرد. چنانچه تعداد خوشه‌ها با تعداد کلاس‌های واقعی یکی باشد، وجود این تعداد لازم خواهد بود اما تعداد بیشتر از آن به این معناست که در حال تقسیم کلاس‌های واقعی هستیم.

در یک نمودار که بهبود واریانس درون خوشه‌ای را نمایش می‌دهد k را به گونه ای برمی‌گزینیم که با انتخاب مقدار بالاتر آن، نتیجه تغییر آنچنانی نداشته باشد.

ب) تعداد خوشه‌ها را از ۱ تا ۱۰ تغییر دهید و الگوریتم را اجرا نمایید. با توجه به روش elbow بهترین تعداد خوشه، برای خوشه‌بندی مجموعه داده را مشخص نمایید و دلیل انتخاب خود را توضیح دهید. نمودار هزینه بر حسب تعداد خوشه را رسم کنید. (برای تابع هزینه می‌توانید از distortion یا inertia استفاده نمایید).

بهترین برابر با ۲ می‌باشد. تا نقطه k=۲ تغییرات مثبت در inertia با شیب تندی حاصل شده است اما پس از آن نتیجه با سرعت کمتری بهبود پیدا می‌کند.

در اینجا برای خوشه‌بندی همچنان از sklearn و K-Means++ استفاده کرده‌ام. همچنین ویژگی‌های Record_ID، Auction_ID و Bidder_ID را حذف کردم.



ج) معیار purity را به ازای تعداد خوشه برابر ۲ (K=۲) محاسبه نمایید.

توضیح آنکه هر خوشه را با کلاسی در نظر می‌گیریم که بیشترین تعداد داده را در خوشه داشته باشد. این تعداد را می‌توان از طریق ماتریس پیشایندی که فرکانس توزیع داده‌ها را نمایش می‌دهد نیز به دست آورد که به این منظور از sklearn استفاده کرده ام.

همچنین می‌توان از طریق شمارش در درون حلقه هم تعدادها را به دست آورد. هر دو راه را در زیر می‌بینید.

مقدار خلوص برابر با ۰/۸۹ می‌باشد که نشاندهنده‌ی تقسیم بندی صحیح داده‌ها تا میزان بالایی می‌باشد.

```
#a function for calculating purity
#I used sklearn here
from sklearn.metrics.cluster import contingency_matrix
def purityscore(truelabels, clusterlabels):
    #contingency matrix
    contmat = contingency_matrix(truelabels, clusterlabels)
    #purity
    purity = np.sum(np.amax(contmat, axis=0))/np.sum(contmat)
    return purity
```

```
kmeans = KMeans(n_clusters = 2).fit(xdf)
```

```
print('Purity: ', purityscore(df['Class'], kmeans.labels_))
```

```
Purity:  0.8932130991931656
```

```
#doing the same thing without sklearn library
tp = list()
for i in range(2):
    current_cluster = df.iloc[kmeans.labels_==i].reset_index(drop=True)
    ans = current_cluster.groupby('Class').count()['Record_ID']
    #the cluster belongs to the class with the most data points in that cluster
    tp.append(max(list(ans)))
purity = sum(tp)/df.shape[0]
print('Purity: ', purity)
```

```
Purity:  0.8932130991931656
```

۲- با استفاده از الگوریتم DBSCAN برای هر یک از مجموعه داده‌های موجود در پوشه‌ی مربوط به این سوال، نمونه ها را همراه با خوشه‌ی نسبت داده شده رسم کنید. به این نکته توجه کنید که داده‌ها می‌توانند متعلق به هیچ خوشه‌ای نباشند و می‌توانند هنگام نمایش به عنوان نویز تلقی شده و نمایش داده شوند. پس از اجرای الگوریتم خوشه‌بندی برای هر یک از مجموعه داده‌ها معیار purity را به دست آورده و به طور کیفی تاثیر نوع مجموعه داده بر کیفیت خوشه‌بندی را مقایسه و تحلیل کنید (در این سوال استفاده از کتابخانه آزاد است).

توابعی برای رسم دو بعدی و سه بعدی داده‌ها و تابعی برای محاسبه‌ی purity تعریف کرده‌ام.

سپس هر دیتاست را خوانده، با آزمون و خطا مقادیری را برای پارامترهای DBSCAN به دست آوردم. البته این آزمون و خطا با توجه به اثر هر یک از این پارامترها بر هر کدام از دیتاست‌ها بود.

دیتاست Compound:

eps = 1.53, min_samples = 5

در دیتاست Compound توزیع داده‌ها در بعضی مناطق تنک و در بعضی متراکم بود که مناطق متراکم حتی با مقدار شعاع کم قابل جداسازی بودند ولی توزیع داده در دو خوشه در بالا سمت راست تنک بوده که موجب می‌شد شعاع کمتر این‌ها را به درستی جدا نکند. به علاوه در کنار این تعداد بالای min-samples موجب می‌شود تعداد بیشتری از داده ها به عنوان داده‌ی پرت تشخیص

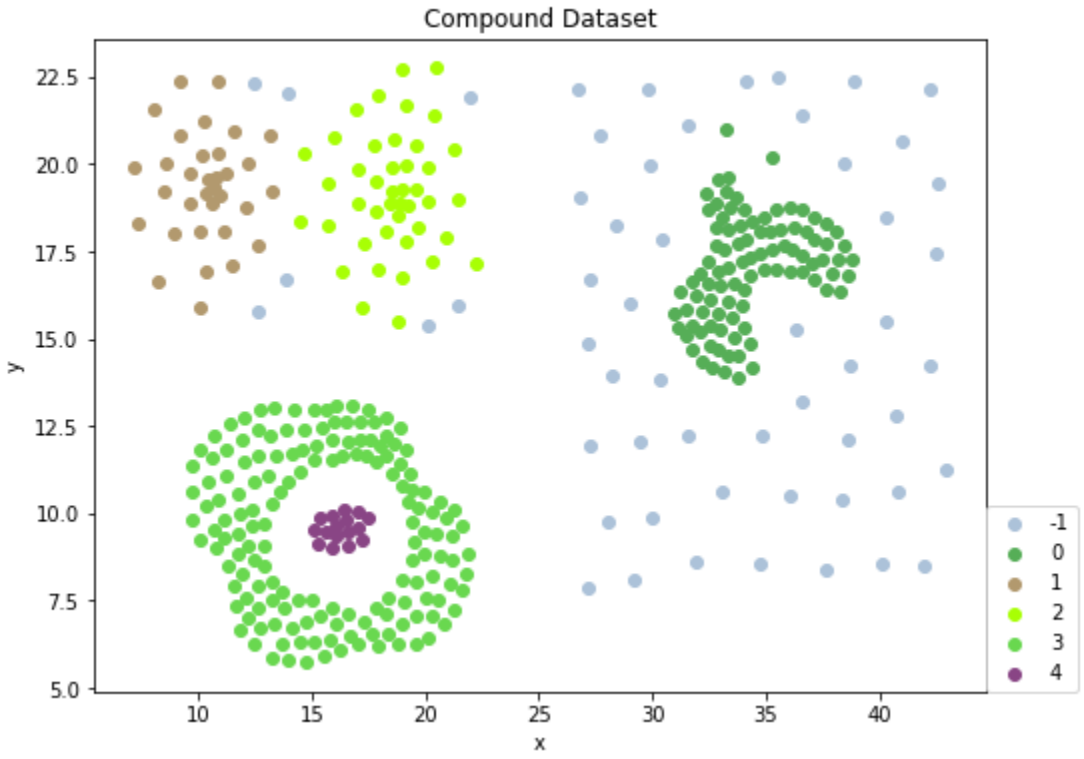
داده شود. از طرفی به این توجه کنید که با شعاع زیاد، دو خوشه‌ی پایین یک خوشه در نظر گرفته می‌شدند. همچنین مقادیر پارامترها باید به گونه‌ای انتخاب می‌شد که داده‌های پراکنده در سمت راست تصویر متعلق به خوشه‌ی میانشان نباشند.

تعداد کلاس‌ها در این دیتاست برابر با ۶ بوده که در زیر نام آنها را مشاهده می‌کنید. تعداد خوشه ها نیز ۵ به علاوه‌ی داده‌های پرت می‌باشد که به عنوان یک خوشه در نظر دارم.

```
Unique Class Labels:  [1 2 3 4 5 6]
```

مقدار خلوص ۰/۹۷ نشان می‌دهد که این خوشه‌بندی تا حد بالایی صحیح است.

```
Purity for Compound Dataset:  0.974937343358396
```



دیتاست D31:

eps = 0.757, min_samples = 20

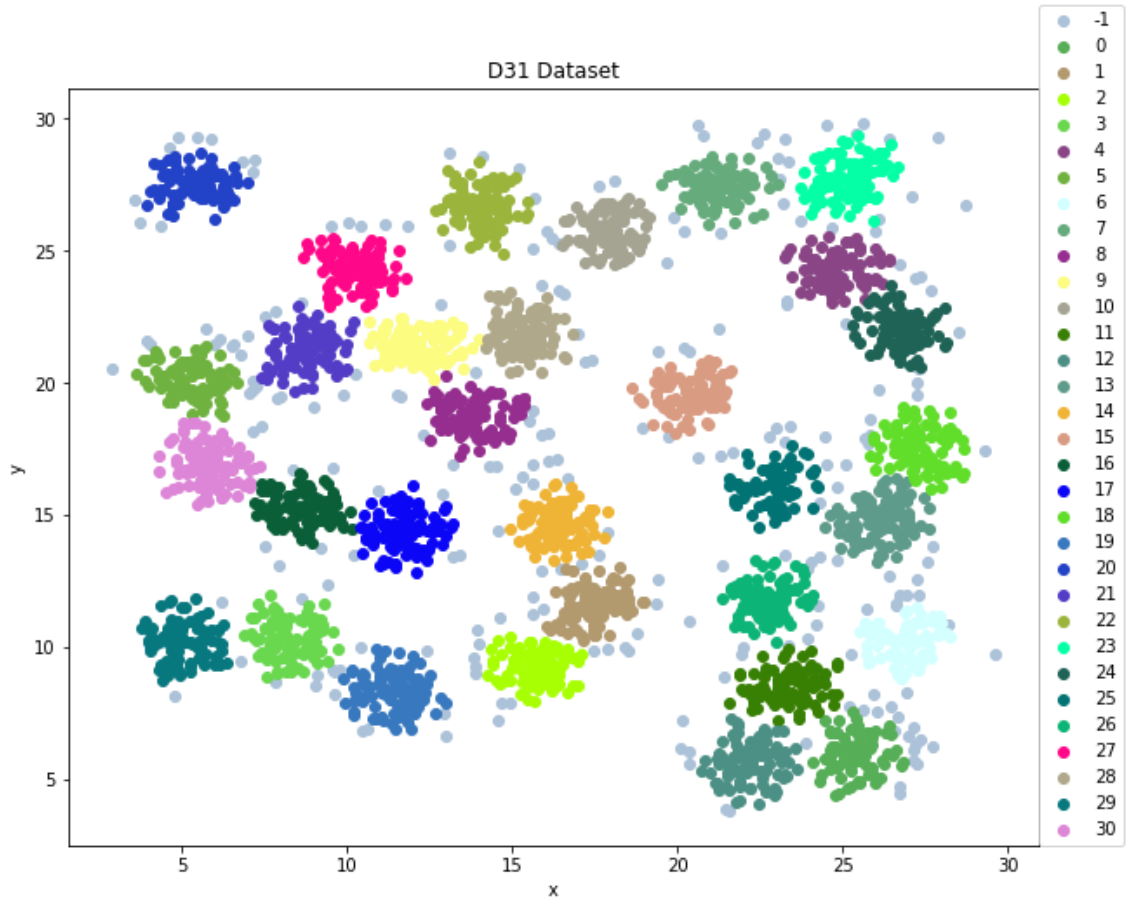
در دیتاست D31 مقدار شعاع لازم بود کوچک باشد؛ چرا که فاصله‌ی کمی بین داده‌های خوشه‌ها وجود داشت. همچنین لازم بود مقدار کمترین نمونه‌ها بیشتر باشد تا برخی داده‌ها در اطراف هر خوشه به عنوان داده‌ی پرت در نظر گرفته شود و خوشه‌هایی که در تصویر می‌بینید به هم متصل نشوند. مقدار زیاد کمترین نمونه‌ها نیز موجب بیشتر شدن تعداد داده‌های پرت می‌شد.

تعداد کلاس‌ها در این دیتاست برابر با ۳۱ بوده که در زیر نام آنها را مشاهده می‌کنید. تعداد خوشه ها نیز ۳۱ به علاوه‌ی داده‌های پرت می‌باشد.

```
Unique Class Labels:  [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31]
```

مقدار خلوص ۰/۹۱ نشان می‌دهد که این خوشه‌بندی تا حد بالایی صحیح است.

```
Purity for D31 Dataset:  0.9109677419354839
```



دیتاست Pathbased:

eps = 2, min_samples = 10

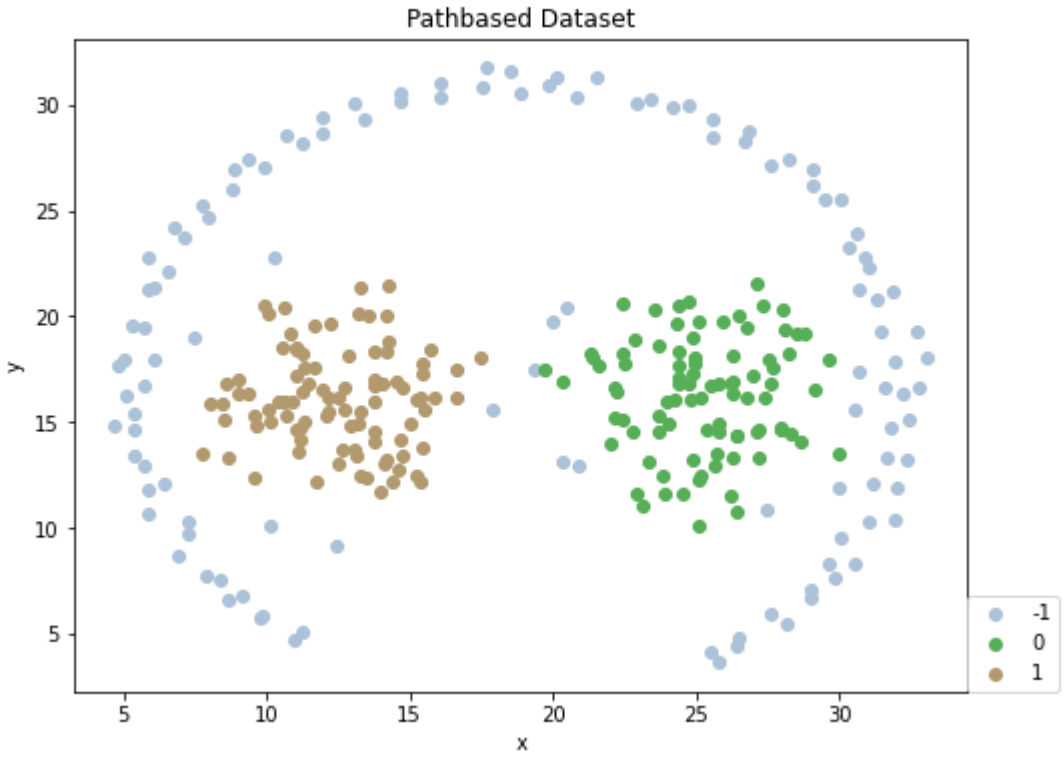
در دیتاست Pathbased به دلیل توزیع تنک داده، لازم دیدم مقدار شعاع را بیشتر بگیرم اما مقدار بزرگتر باعث ایجاد یک خوشه‌ی واحد می‌شد. پس از این، مقدار کم برای کمترین نمونه، موجب می‌شد خوشه‌های کوچکی تشکیل شوند که همگی به یک کلاس تعلق داشتند. بنابراین در نهایت مقادیری را انتخاب کردم که داده‌هایی را پرت در نظر می‌گرفت اما همین داده‌های پرت در واقع از یک کلاس بودند. به عبارتی توزیع به گونه‌ای بود که DBSCAN داده‌های یکی از کلاس‌ها را پرت در نظر می‌گرفت؛ چرا که به دلیل جداسازی صحیح دو کلاس دیگر نیاز بود مقادیر به گونه‌ای تعریف شود که با داده‌های کلاس دیگر همخوانی نداشت.

تعداد کلاس‌ها در این دیتاست برابر با ۳ بوده که در زیر نام آنها را مشاهده می‌کنید. تعداد خوشه ها نیز ۲ به علاوه‌ی داده‌های پرت می‌باشد که به عنوان یک خوشه در نظر دارم.

```
Unique Class Labels:  [1 2 3]
```

مقدار خلوص ۰/۹۶ نشان می‌دهد که این خوشه‌بندی تا حد بالایی صحیح است.

```
Purity for Pathbased Dataset:  0.96
```



دیتاست Rings:

eps = 5, min_samples = 8

برای دیتاست Rings لازم بود مقدار شعاع را بیشتر بگیرم تا داده‌ها بتوانند به یکدیگر دسترسی پیدا کنند. همچنین مقدار کمترین نمونه‌ها با انتخابی که برای شعاع داشتم، با کمتر شدن موجب تشکیل خوشه‌های کوچک می‌شد و مقدار بزرگتر آن می‌توانست داده‌های پرت ایجاد کند.

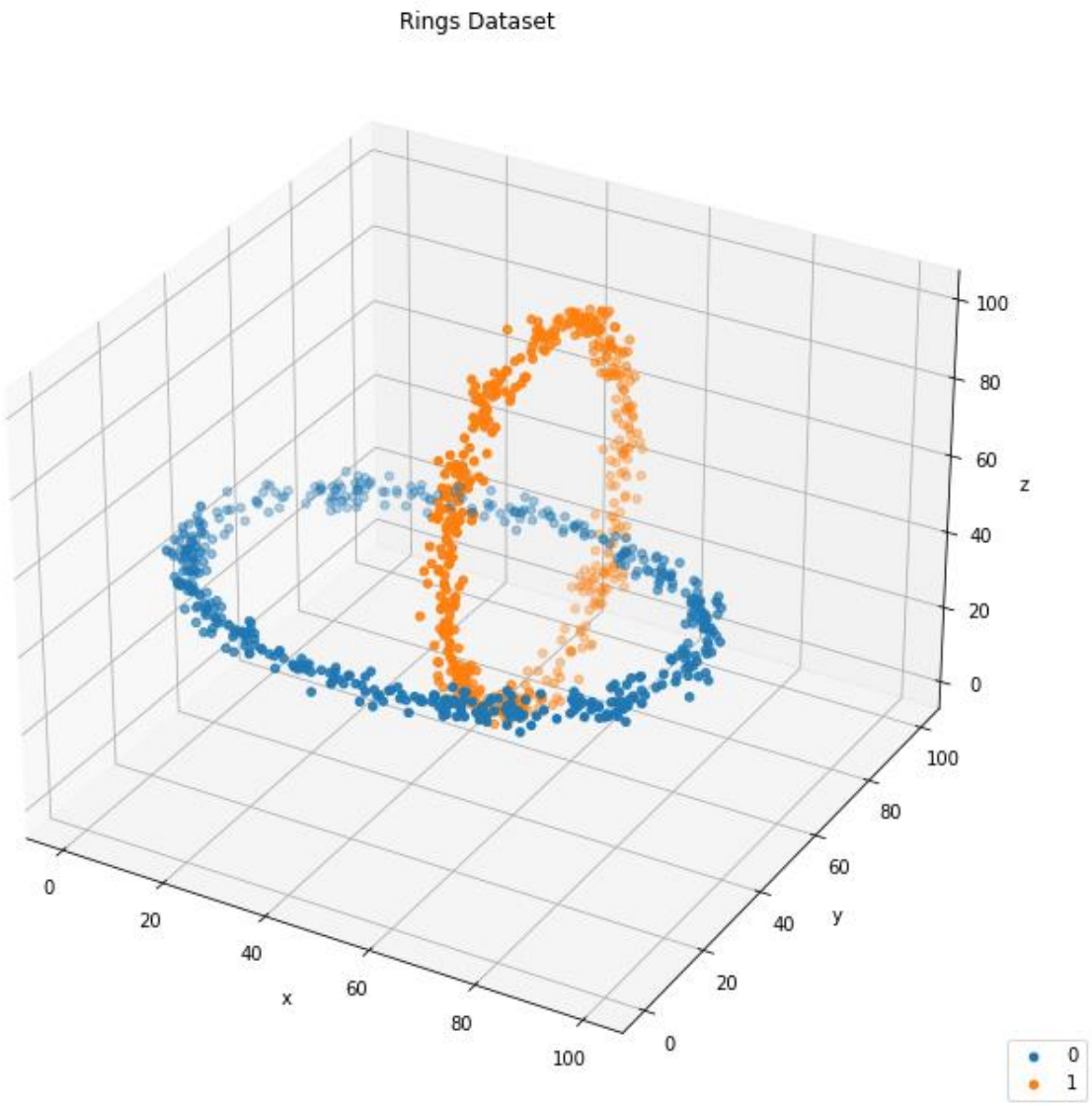
این دیتاست سه ویژگی داشت که برای رسم از تابع رسم سه بعدی استفاده کردم.

تعداد کلاس‌ها در این دیتاست برابر با ۲ بوده که در زیر نام آنها را مشاهده می‌کنید. تعداد خوشه ها نیز ۲ تاست.

```
Unique Class Labels:  [1 2]
```

مقدار خلوص ۱ نشان می‌دهد که این خوشه‌بندی کاملا صحیح است.

```
Purity for Rings Dataset:  1.0
```



دیتاست Spiral:

eps = 1.5, min_samples = 3

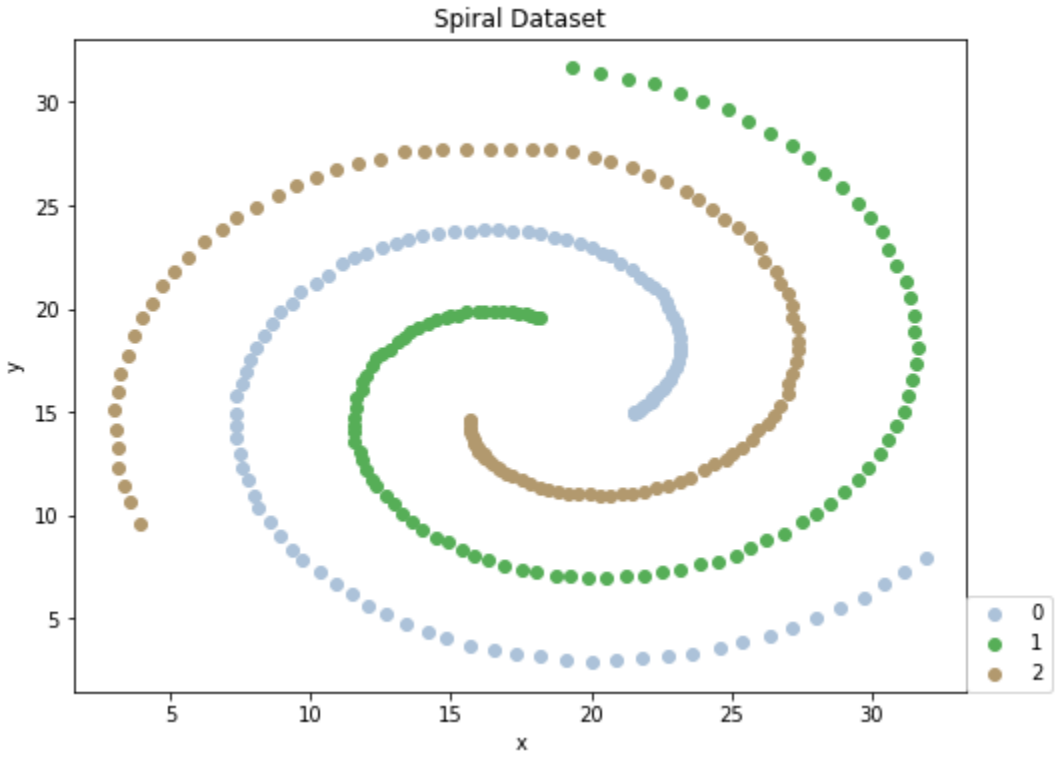
برای دیتاست Spiral لازم بود مقدار شعاع را در حدی بگیرم که از فاصله‌ی بین هر کلاس از داده‌ها کمتر باشد درحالیکه داده‌ها بتوانند داده‌های درون کلاس خود را بیابند. تعداد کمترین نمونه‌ها نیز لازم بود کم باشد تا داده‌های نوک هر کدام از آن خطوط Spiral (به طور کلی برخی بخش ها که تعدادی نقطه، کمی بیشتر از بقیه فاصله دارند) به عنوان داده‌های پرت در نظر گرفته نشوند.

تعداد کلاس‌ها در این دیتاست برابر با ۳ بوده که در زیر نام آنها را مشاهده می‌کنید. تعداد خوشه ها نیز ۳ تاست.

```
Unique Class Labels:  [1 2 3]
```

مقدار خلوص ۱ نشان می‌دهد که این خوشه‌بندی کاملا صحیح است.

```
Purity for Spiral Dataset:  1.0
```



به نظر می‌رسد هر چه تراکم داده‌ها درون کلاس‌ها بیشتر باشد و همچنین داده‌های کلاس‌ها در هم فرورفتگی نداشته باشند نتیجه بهتر است. بعلاوه توزیع‌های مشابه کلاس‌ها موجب می‌شود تشخیص بسیار بهتر شود (با توجه به دیتاست Compound و Pathbased).

۳- در این بخش می‌خواهیم دو الگوریتم value iteration و policy iteration را برای محیط Frozen Lake مانند شکل زیر پیاده‌سازی نماییم.

در این محیط عامل با شروع حرکت از خانه‌ی شروع (S) می‌خواند به خانه‌ی هدف (G) برسد. در این بین خانه‌های یخ زده‌ای (F) هم وجود دارد. همچنین گودال‌هایی (H) نیز در نقشه (محیط) موجود است. عامل باید از طریق خانه‌های

یخزده حرکت کرده و به خانه‌ی هدف برسد. توجه کنید که عامل به هدف محیط شامل احتمال گذار وضعیت‌ها و میزان پاداش دسترسی دارد. در این بخش قصد داریم سیاست بهینه را برای محیط ۵ در ۵ شکل بالا به دست آوریم. هنگامی که عامل به خانه‌ی G یا H برسد یک اپیزود تمام می‌شود. در صورتی که عامل در خانه G قرار بگیرد، پاداش ۱+ می‌گیرد. در مابقی موارد عامل پاداشی دریافت نمی‌کند. در پیاده‌سازی الگوریتم‌ها $\gamma = 0.85$ در نظر بگیرید (انتخاب شرط خاتمه‌ی مناسب به عهده‌ی شما می‌باشد).

الف) الگوریتم value iteration را پیاده‌سازی کرده و مقادیر V^* را به دست آورید. زمان اجرا و تعداد تکرار مورد نیاز را نمایش دهید. سیاست بهینه را به دست آورید و آن را به صورت یک جدول متشکل از حروف U (بالا)، R (راست)، D (پایین) و L (چپ) نمایش دهید.

```python
def value_iter(env, gamma, theta):
    # env.env.nS is the number of states
    # env.env.nA is the number of actions
    # env.env.P is the model that includes:
    # p: probability of going from one state to another via a specific action
    # r: reward
    # s_: next state
    V = np.zeros(env.env.nS) #Initial Values
    pi = np.zeros(env.env.nS) #Initial Policy
    count_iter = 0 # for Counting Iteration
    while True:
        count_iter+=1
        delta = 0
        for s in range(env.env.nS): #for each state compute:
            v = V[s]
            values = list()
            for a in range(env.env.nA): #for each action compute
                val = 0
                for p, s_, r, _ in env.env.P[s][a]:
                    val += p * (r + gamma * V[s_])#value that gets added if a specific action is taken in a specific sta
                values.append(val)
            V[s] = max(values)#Figuring out the best value for this state
            pi[s] = np.argmax(values)#Figuring out which action gives the best value
            delta = max(delta, abs(v - V[s]))#Checking how much the value changes
        if delta < theta: break#If there's not much change in value, stop

    return V, pi, count_iter
```

در این کد به ازای هر state و action مقدار value را با استفاده از فرمول (۱) محاسبه کردم. سپس بیشترین مقدار value به ازای هر یک از action ها را در V^* قرار دادم و action منتخب را نیز در pi ذخیره کردم. این عمل تا زمانی که تغییر آنچنانی در V ایجاد نشود ادامه دارد.

$$v_{i+1}(s) \leftarrow \max_a \sum_{s',r} p\,(s', r|s, a)[r + \gamma v_i(s')] \qquad (۱)$$

نتایج را به ازای is_slippery = True در زیر می‌بینید.

```
Value Iteration
[[0.00698058 0.00750863 0.00350883 0.00138496 0.        ]
 [0.01021292 0.01204922 0.         0.         0.10575782]
 [0.01383751 0.02481905 0.06172147 0.09328092 0.26750555]
 [0.         0.         0.09976105 0.         0.57087423]
 [0.10099983 0.15470592 0.29041378 0.57992437 0.        ]]
Time:  0.03497815132141113
Number  of Iterations:  27
Policy:
 [['D' 'L' 'U' 'L' 'L']
 ['D' 'L' 'L' 'L' 'D']
 ['U' 'U' 'D' 'D' 'R']
 ['L' 'L' 'L' 'L' 'R']
 ['D' 'D' 'D' 'D' 'L']]
```

ب) الگوریتم policy iteration را نیز مانند حالت قبل پیاده‌سازی کرده و زمان اجرا و تعداد تکرار آن را با مورد قبل مقایسه کنید. سیاست بهینه را مانند قسمت قبل نمایش دهید.

```python
V = np.zeros(env.env.nS) #Initial Values
#pi = np.random.randint(4,size=env.env.nS) #Random Initial Policies
pi = np.ones(env.env.nS) #Initial Policies (all of them are set to one)
count_iter = 0# for Counting Iteration
while True:
    count_iter+=1
    #Compute Values for this Policy
    while True:
        delta = 0
        for s in range(env.env.nS):#for each state compute:
            v = V[s]
            val = 0
            for p, s_, r, _ in env.env.P[s][pi[s]]:#compute only for action 1 (down)
                val += p * (r + gamma * V[s_])
            V[s] = val
            delta = max(delta, abs(v - V[s]))#Checking how much the value changes
        if delta < theta: break#If there's not much change in value, stop
    #Update Policy
    policy_stable = True #Assuming we've already got the best policy
    for s in range(env.env.nS):#for each state compute:
        old_action = pi[s]
        values = list()
        for a in range(env.env.nA):#for each action compute:
            val = 0
            for p, s_, r, _ in env.env.P[s][a]:
                val += p * (r + gamma * V[s_])#value that gets added if a specific action is taken in a specific sta
            values.append(val)#Figuring out the best value for this state
        pi[s] = np.argmax(values)#Figuring out which action gives the best value
        if old_action != pi[s]: policy_stable = False #If It's not the old action keep updating
    if policy_stable: break#If the policy has stablized, stop updating
```

در اینجا در ابتدا تمامی action ها را به سمت پایین در نظر گرفتم (در حقیقت حرکتی تصادفی می‌توان برگزید، در اینجا برای ثبات پاسخ و همچنین چون هدف در پایین صفحه‌ی شطرنجی قرار دارد، تمامی را حرکت شماره‌ی یک یا به عبارتی پایین در نظر گرفتم) و مقادیر Value را بر اساس این action برای تمام state ها محاسبه کردم. این عمل تا زمانی ادامه دارد که تفاوت چندانی در مقادیر V ایجاد نشود.

سپس در بخش Update Policy به دنبال جهتی می‌گردیم که Value بهتری را حاصل دهد. از آن جهت که یافتن Policy مناسب حداکثر به اندازه‌ی تکرار Value Iteration طول می‌کشد، ممکن است با انتخابی تصادفی برای Policy اولیه، این تعداد

تکرار کمتر هم شود. به خصوص در مواقعی که سطح لیز (is_slippery = True) است؛ چرا که در این صورت تعداد حالاتی که Agent ممکن است برود بیشتر می‌شود.

همانطور که در نتایج می‌بینید تعداد تکرار و زمان اجرا هر دو کمتر شده است.

نتایج را به ازای is_slippery = True در زیر می‌بینید.

```
Policy Iteration
[[0.00688123 0.00742604 0.00345214 0.0013585  0.        ]
 [0.01015866 0.01200206 0.         0.         0.10575116]
 [0.01379926 0.02479848 0.06173264 0.09328062 0.26749831]
 [0.         0.         0.09980908 0.         0.5708703 ]
 [0.10128224 0.15490417 0.29053499 0.57997889 0.        ]]
Time:  0.019987821578979492
Number  of Iterations:  3
Policy:
 [['D' 'L' 'U' 'L' 'L']
 ['D' 'L' 'L' 'L' 'D']
 ['U' 'U' 'D' 'D' 'R']
 ['L' 'L' 'L' 'L' 'R']
 ['D' 'D' 'D' 'D' 'L']]
```

۴- مجموعه داده‌ی SeoulBikeData.csv در فایل مجموعه داده‌ها قرار داده شده است. با استفاده از آن موارد زیر را انجام دهید (استفاده از کتابخانه در تمامی بخش‌های سوال مجاز است).

الف) پیش‌پردازش‌های لازم را انجام دهید.

```
df = df.replace({'Seasons': {'Spring': 0, 'Summer': 1, 'Autumn': 2, 'Winter': 3 }
                , 'Holiday': {'No Holiday': 0, 'Holiday': 1}
                , 'Functioning Day': {'Yes': 1, 'No': 0}})
df = df.drop(columns=['Date'])
df.head()
```

در تصویر می‌بینید که تمامی ویژگی‌هایی که مقدار غیر عددی داشتند، به عددی تبدیل کردم.

همچنین از K-Fold Cross Validation در تمامی بخش‌های این سوال استفاده کردم.

لازم به ذکر است برای نرمال سازی داده در خود مدل‌های یادگیرنده، پارامتر normalize را برابر با True قرار دادم.

ب) همبستگی بین ویژگی‌ها را استخراج کرده و با توجه به آن بهترین ویژگی‌ها را انتخاب کنید. در این مرحله شما باید تعداد ویژگی‌های انتخاب شده را با توجه به یک مدل رگرسیون خطی پایه مورد بررسی قرار داده و بهترین K را پیدا کنید.

مقدار score میانگین را برای مدل آموزش دیده بر دیتاست بدون حذف ویژگی به دست آورده و در متغیری ذخیره می‌کنم.

در ابتدا قدر مطلق همبستگی بین ویژگی‌ها را محاسبه کردم و مقادیر قطر (که همبستگی هر ویژگی با خودش است) را برابر صفر قرار دادم تا در روشی که در پیش گرفتم اخلالی پیش نیاید. مقدار threshold را برابر ۰/۹ قرار دادم.

در یک حلقه بزرگترین مقدار همبستگی را با threshold مقایسه می کنم و در صورتی که ویژگی‌هایی با همبستگی بیشتر از آن وجود داشت، به این ترتیب عمل می کنم: بین این دو ویژگی، آن که همبستگی کمتری با نتیجه دارد حذف کرده و سپس با استفاده از K-Fold Cross Validation مدل رگرسیون خطی را آموزش داده و میانگین score مدل‌ها را به دست می‌آورم. اگر این score از آنچه بدون حذف ویژگی کنونی به دست آورده بودم بهتر بود، به کار ادامه می‌دهم، در غیر این صورت از حلقه خارج شده و تعداد ویژگی‌های حذف شده را می‌نویسم.

نتیجه بدون حذف ویژگی:

```
Score without Removing any Features:  0.5244805851401724
```

ویژگی‌های با همبستگی بالاتر از ۰/۹:

```
Featurs with correlation more than Threshold:
 Temperature ,  Dew point temperature
```

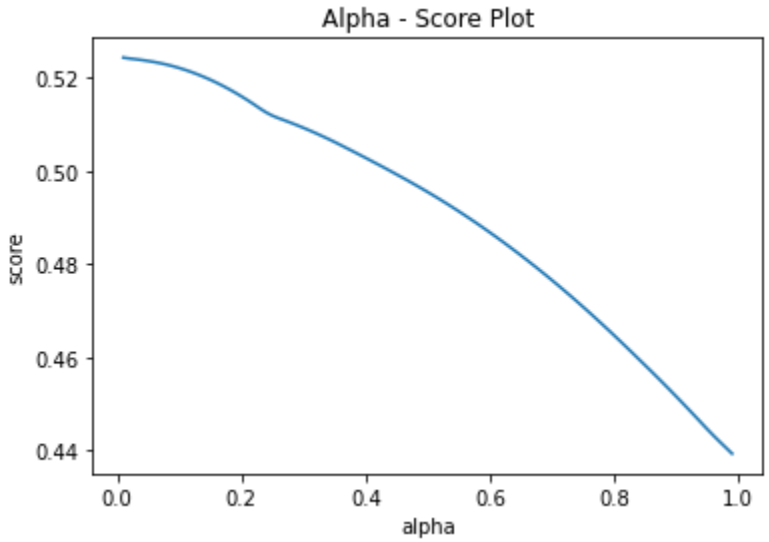
نتیجه با حذف Dew Point Temperature:

```
Removing  Dew point temperature
Score After Feature Removal:  0.5242881105793147
```

بنابراین هیچ ویژگی حذف نمی‌شود و حلقه پایان می‌یابد.

```
Number of Features Removed:  0
```

ج) با استفاده از داده پیش‌پردازش‌شده مدل رگرسیون لسو را آموزش دهید. نقش پارامتر α را در این مدل بررسی کرده و با جستجو، بهترین مقدار آن را به دست آورید.



مقدار α را از ۰/۰۱ تا ۱ با فاصله‌ی ۰/۰۱ در نظر گرفتم. سپس با K-Fold Cross Validation مقدار score میانگین را به دست آوردم. مقدار ۰/۰۱ برای α بهترین نتیجه را می‌داد.

در نهایت مقادیر Coefficient را در هر اجرای درون K-Fold Cross Validation چاپ کردم. همانطور که می‌بینید هیچ یک صفر نشده‌اند.

```
Lasso Model Coefficients:
 [ 2.72886523e+01  2.45730311e+01 -8.46643623e+00  5.92915221e+00
  4.54599764e-02  6.08237737e+00 -7.45128895e+01 -5.74089383e+01
  1.26231448e+01 -2.83917404e+01 -1.27940428e+02  7.97820538e+02]
Lasso Score 0.5143528693010182
Lasso Model Coefficients:
 [ 2.73534633e+01  2.67348617e+01 -8.53560469e+00  2.30236843e+00
  3.99017035e-02  4.14328216e+00 -7.75499385e+01 -5.65530881e+01
  2.04403485e+01 -2.74082101e+01 -1.61687048e+02  7.92989331e+02]
Lasso Score 0.5272464875156628
Lasso Model Coefficients:
 [ 2.75438849e+01  2.68209808e+01 -8.01405220e+00  3.82679252e+00
  3.86450549e-02  3.84750695e+00 -7.34555816e+01 -6.01565386e+01
  1.63166669e+01 -2.47356145e+01 -1.41647653e+02  7.97518113e+02]
Lasso Score 0.5321045843840237
Lasso Model Coefficients:
 [ 2.78269438e+01  2.67192661e+01 -7.70456806e+00  6.38887214e+00
  3.93095495e-02  3.64848917e+00 -7.27217171e+01 -6.13040534e+01
  1.57567761e+01 -2.59316527e+01 -1.28233661e+02  7.95808615e+02]
Lasso Score 0.5244583777703502
Lasso Model Coefficients:
 [ 2.72822758e+01  2.74366429e+01 -8.00292694e+00  3.35921307e+00
  3.86015225e-02  3.08936466e+00 -7.88755701e+01 -6.11604513e+01
  1.59136086e+01 -3.00694673e+01 -8.57663640e+01  7.85700466e+02]
Lasso Score 0.5239081435235574
Average Score:  0.5244140924989226
```

هر چه پارامتر α بزرگتر باشد، ضرایب بیشتری از ویژگی‌ها صفر می‌شوند. این پارامتر برای رگولاریزیشن استفاده می‌شود اما به دلیل آنکه در مجموع قدرمطلق وزن ویژگی‌ها ضرب شده موجب صفر شدن وزن ویژگی‌های با تاثیر کمتر می‌شود. رگرسیون لاسو به همین دلیل به نوعی انتخاب ویژگی نیز می‌کند. در اینجا مشخص است حذف ویژگی به نفع ما نیست.

د) ویژگی‌های انتخاب شده در بخش ب و ج را با هم مقایسه کنید. چه نتیجه‌ای می‌گیرید؟

در هر دو هیچ یک از ویژگی‌ها حذف نمی‌شود. با وجود همبستگی زیاد این دو ویژگی به یکدیگر، این دو به ترتیب بیشترین و سومین بیشترین همبستگی را با خروجی دارند. درحالیکه باقی ویژگی‌ها به اندازه‌ی اینها با خروجی همبستگی ندارند. این مسئله موجب می‌شود هر دو ویژگی حفظ شوند.

ه) برای بهبود عملکرد مدل چه پیشنهادی دارید؟

مسئله اینجاست که بیشتر از آنکه ویژگی‌ها با کلاس‌ها همبستگی داشته باشند، با یکدیگر همبستگی دارند. نیاز به انتخاب ویژگی‌های بهتر و یا استفاده از روش‌های Feature Extraction می‌باشد.

۵- مجموعه داده‌ی heart_failure_clinical_records_dataset.csv را بارگذاری کنید. شما باید با استفاده از ویژگی‌های موجود، هر فرد را بر اساس مقادیر ستون DEATH_EVENT دسته‌بندی کنید (استفاده از کتابخانه در تمامی بخش‌های سوال مجاز است).

الف) پیش‌پردازش‌های لازم را انجام دهید. این مجموعه داده شامل مقادیر گم شده است. روش‌های مختلفی برای برطرف کردن این مشکل پیشنهاد شده است. درباره‌ی آن‌ها تحقیق کرده و با ذکر دلیل یکی از این روش‌ها را انتخاب کرده و مقادیر گم شده مجموعه داده را برطرف کنید.

در ابتدا بررسی کردم در چه جاهایی مقادیر گم شده داریم.

در کل ۳۰۰ نمونه شامل ۹۷ تا از کلاس یک و ۲۰۳ تا از کلاس صفر داریم. از میان اینها ۱۴ نمونه، ۸ تا از کلاس یک و ۶ تا از کلاس صفر شامل مقادیر گم شده بود. در مجموع از ۳۹۰۰ عدد، ۳۲ مقدار گم شده داشتیم. با توجه به اینکه داده نامتوازن و تعداد نمونه‌های کلاس یک کمتر بود، تصمیم گرفتم نمونه ای را حذف نکنم. یکی از نمونه‌ها ۷ مقدار گم شده (حدود نصف ویژگی‌ها) داشت که حذف این نمونه در روشی که برای پر کردن مقادیر انتخاب کردم تاثیر منفی می‌گذاشت پس آن را نگه داشتم.

دو راه بهترین نتایج را می‌دادند. یکی استفاده از Multivariate Feature Imputation و دیگری Hot Deck Imputation بود. مسئله این بود که روش اول به دلیل استفاده از رگرسیون مقادیر غیر صفر و یک برای ویژگی‌هایی که تنها این مقادیر را داشتند اختصاص می‌داد. پس تصمیم گرفتم راه دوم را پیش بگیرم. این راه در کتابخانه‌ای موجود نیست ولی رفتار مشابه آن همان پر کردن مقادیر گم شده، با مقدار نمونه‌ی پیشین یا پسین خود می‌باشد. البته این رفتار مشابه، کمی از حالت تصادفی که در Hot Deck Imputation است می‌کاهد. برای اینکه این مقداردهی مناسب‌تر صورت بگیرد، نمونه‌ها را به ترتیب کلاس‌هایشان مرتب کردم. سپس با استفاده از ffill کتابخانه‌ی pandas مقادیر گم شده را پر کردم. علت دیگر انتخاب من این بود که نمونه‌های دارای مقادیر گم شده پشت سر هم نیستند و بینشان فاصله است بنابراین نمونه‌های تازه مقداردهی شده شباهت به یک نمونه‌ی خاص پیدا نمی‌کنند.

در تمامی روش‌هایی که امتحان کردم، بهترین نتیجه را همین راه داشت.

ب) بهترین K ویژگی را با توجه به اهمیت آن‌ها انتخاب کنید. همانند بخش ب سوال ۴، باید تعداد ویژگی‌های انتخاب شده را با توجه به یک مدل دسته‌بند پایه مورد بررسی قرار دهید و بهترین K را پیدا کنید.

برای این کار از Logistic Regression به همراه K-Fold Cross Validation (با K=۵) استفاده کردم. با انتخاب threshold برابر با ۰/۹ هیچ یک از ویژگی‌ها به این علت که همبستگی بالاتر از این مقدار ندارند، حذف نخواهد شد. اگر این مقدار threshold را پایین‌تر انتخاب کنیم (صرفا به علت نمایش کارکرد آن) مدل Logistic Regression با K-Fold Cross Validation مقدار score کمتری خواهد داشت بنابراین در این سوال نیز هیچ ویژگی حذف نمی‌شود (البته لازم به ذکر است بدون استفاده از K-Fold Cross Validation گاهی تنها یک ویژگی sex با threshold = 0.4 حذف می‌شد).

```
Score without Removing any Features:  0.8133333333333332

Featurs with correlation more than Threshold:
 sex ,  smoking
Removing  sex
Score After Feature Removal:  0.8
Number of Features Removed:  0
```

ج) ۳ مدل مختلف رای‌گیری که هر کدام شامل ۳ دسته‌بند است را برای بهترین K ویژگی آموزش دهید و بهترین مدل را انتخاب کنید.

برای راحتی کار K-Fold Cross Validation را در یک تابع انجام می‌دهم که دسته‌بند را دریافت می‌کند و score را برمی‌گرداند.

۴ مدل مختلف را امتحان کردم.

مدل soft voting با ۳ مدل SVM با نتیجه‌ی score حدودا بین ۷۹ تا ۸۲ درصد،

```
#Using three SVMs
svm1 = svm.SVC(probability=True, kernel='poly', degree=1)
svm2 = svm.SVC(probability=True, kernel='poly', degree=2)
svm3 = svm.SVC(probability=True, kernel='poly', degree=3)

eclf = VotingClassifier(estimators=[('svm1', svm1), ('svm2', svm2), ('svm3', svm3)],voting='soft')
score = kfoldscore(eclf)

print(score)
```

```
0.79
```

مدل hard voting با ۳ مدل K-NN با نتیجه‌ی score حدودا بین ۶۹ تا ۷۱ درصد،

```
# Using three KNNs
knn1 = KNeighborsClassifier(n_neighbors=1)
knn2 = KNeighborsClassifier(n_neighbors=3)
knn3 = KNeighborsClassifier(n_neighbors=5)
eclf = VotingClassifier(estimators=[('knn1', knn1), ('knn2', knn2), ('knn3', knn3)],voting='hard')
score = kfoldscore(eclf)

print(score)
```

```
0.6933333333333334
```

مدل soft voting با ۳ مدل Logistic Regression، Decision Tree و Gaussian Naive Bayes با نتیجه‌ی score حدودا بین ۷۹ تا ۸۳ درصد،

```
# Using Logistic Regression, Decision Tree and GaussianNB
clf1 = LogisticRegression(random_state=1)
clf2 = DecisionTreeClassifier(max_depth = 3,min_samples_split=30,max_features=4)
clf3 = GaussianNB()

eclf = VotingClassifier(estimators=[('lr', clf1), ('dt', clf2), ('gnb', clf3)],voting='soft')
score = kfoldscore(eclf)

print(score)
```

```
0.8033333333333333
```

و بهترین مدل، مدل soft voting با ۳ درخت تصمیم گیری با نتیجه‌ی score حدودا بین ۷۹ تا ۸۴ درصد.

```
# Using three Decision Trees
dt1 = DecisionTreeClassifier(max_depth = 3,min_samples_split=10,max_features=4)
dt2 = DecisionTreeClassifier(max_depth = 3,min_samples_split=30,max_features=4)
dt3 = DecisionTreeClassifier(max_depth = 3,min_samples_split=50,max_features=4)
eclf = VotingClassifier(estimators=[('dt1', dt1), ('dt2', dt2), ('dt3', dt3)],voting='soft')
score = kfoldscore(eclf)

print(score)
```

```
0.8233333333333333
```

د) دسته‌بندهای مورد استفاده در بهترین مدل را با استفاده از داده‌های به دست آمده در بخش ب (بهترین K ویژگی) به صورت مجزا آموزش دهید. از مقایسه عملکرد دسته‌بندها به صورت تکی و گروهی چه نتیجه‌ای می‌گیرید؟

مدل رای‌گیری گاهی ممکن است به خوبی یکی از مدل‌های پایه‌ی خود عمل نکند. بهترین مدل ما اینگونه نیست. نتایج را برای دسته‌بندهای پایه‌ی آن مشاهده می‌کنید.

در تمامی اجراهایی که داشتم، این مدل بهتر از دسته‌بندهای پایه‌ی خود عمل می‌کرد.

```
# Decision Tree 1
score = kfoldscore(dt1)
print('Decision Tree 1 Score: ', score)
```

```
Decision Tree 1 Score:  0.7766666666666667
```

```
# Decision Tree 2
score = kfoldscore(dt2)
print('Decision Tree 2 Score: ',score)
```

```
Decision Tree 2 Score:  0.7166666666666666
```

```
# Decision Tree 3
score = kfoldscore(dt3)
print('Decision Tree 3 Score: ',score)
```

```
Decision Tree 3 Score:  0.7933333333333332
```

می‌توان نتیجه گرفت با استفاده از رای‌گیری بین چند دسته‌بند ضعیف به نتیجه بهتری رسید و وقتی این دسته‌بندها همگی ضعیف عمل می‌کنند و با یکدیگر روی تشخیص کلاس نمونه توافق آنچنانی ندارند، می‌توانند نتیجه‌ی نهایی را بهتر کنند.