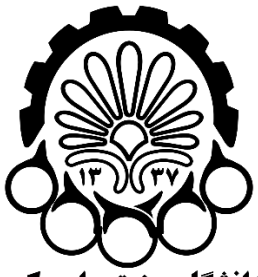



دانشگاه صنعتی امیرکبیر
( پلی تکنیک تهران )

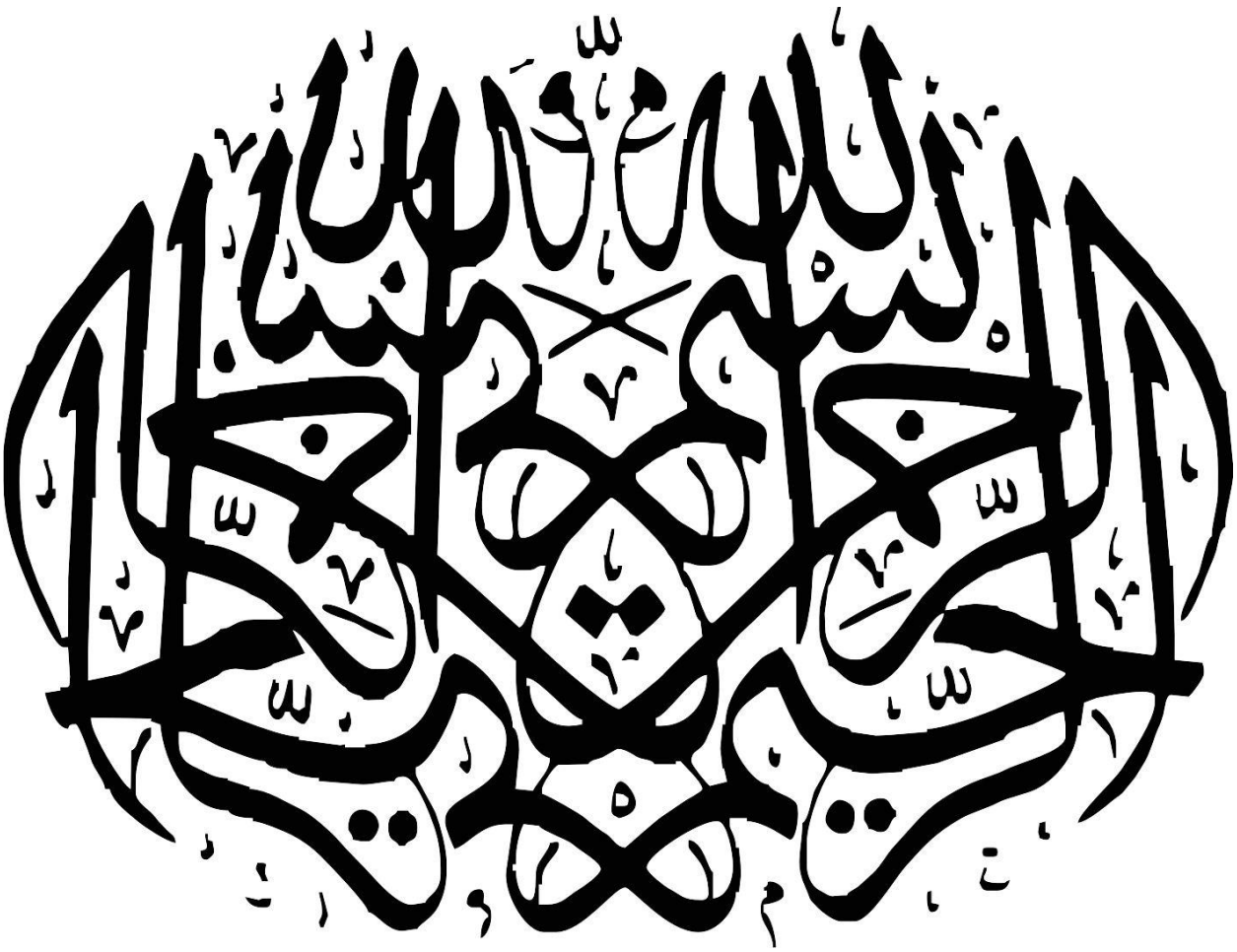
# گزارش تکلیف سوم درس داده‌کاوی

## استاد درس: دکتر احمدی

نیما محمودیان ۴۰۲۱۲۵۰۰۵

محسن فراغه ۴۰۱۲۲۵۰۰۲

معصومه بهبهانی زاده ۴۰۲۱۹۱۰۰۳



# فهرست مطالب



# فهرست شکل‌ها

## ۱-پاسخ سوال اول

ابتدا داده‌ها را با روشهای سلسله مراتبی ادغامی و خوشه بندی کامیانگین در دو خوشه قرار- دهید. عملکرد روشها را با معیارهای مختلف ارزیابی کنید.
برای این منظور، ابتدا داده‌ها را با دستور pd.read_csv فراخوانی می‌کنیم و در متغیر df ذخیره می‌کنیم. داده‌های دریافت شده، از قبل پیش‌پردازش شده‌اند و برای خوشه‌بندی آماده هستند.

حال تمامی ستون‌های دیتافریم به جز ستون هدف را به شکل یک آرایه نامپای در می‌آوریم و در متغیر X ذخیره می‌کنیم. شکل (۱) نحوه انجام این کار را نمایش می‌دهد.

```
X = np.array(df.drop("Diagnosis", axis=1))
✓ 0.0s
```

شکل ۱: آماده سازی داده‌ها برای خوشه‌بندی

### ۱-۱-خوشه‌بندی سلسله‌مراتبی ادغامی

برای انجام خوشه‌بندی با این روش از نمایش داده شده در شکل (۲) استفاده می‌کنیم.

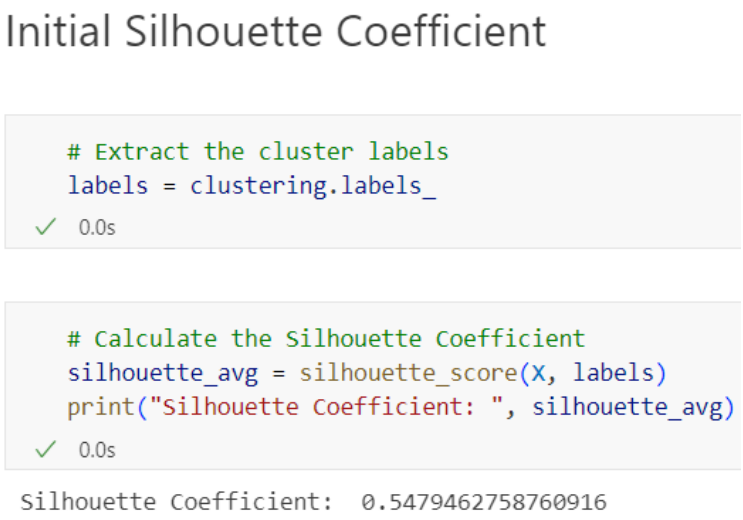
```
# Instantiate the model
clustering = AgglomerativeClustering(linkage='ward', n_clusters=2)
# Fit model
clustering.fit(X)
✓ 0.0s
```

```
▾ AgglomerativeClustering
AgglomerativeClustering()
```

شکل ۲: کد خوشه‌بندی سلسله‌مراتبی ادغامی

این کد یک شی از کلاس AgglomerativeClustering با استفاده از معیار linkage به روش 'ward' و تعداد خوشه‌ها برابر با ۲ ایجاد می‌کند. سپس مدل خوشه‌بندی با داده‌های ویژگی‌ها (X) برازش داده می‌شود. متد fit(). خوشه‌بندی داده‌ها را انجام می‌دهد و هر نمونه داده را به یکی از دو خوشه تخصیص می‌دهد. حال یک ارزیابی اولیه از خوشه‌بندی انجام می‌دهیم. برای این منظور از معیار سیلوئت استفاده می‌کنیم.

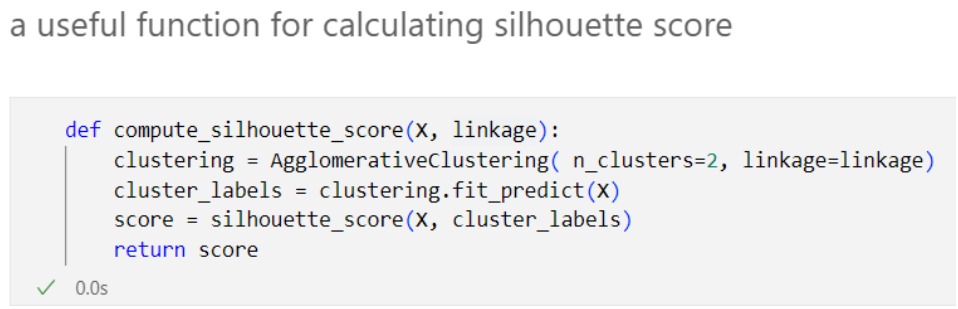
شکل (۳) نحوه ارزیابی را نمایش می‌دهد. ابتدا برچسب‌های خوشه‌ای که هر نمونه داده به آن تخصیص یافته است از مدل استخراج و در متغیری به نام labels ذخیره می‌شود. این برچسب‌ها نشان‌دهنده خوشه‌ای هستند که هر نمونه به آن تعلق دارد. سپس ضریب سیلوئت با استفاده از تابع silhouette_score محاسبه می‌شود. این ضریب، میانگین نزدیکی نمونه‌ها به خوشه‌های خودی را نسبت به نزدیک‌ترین خوشه دیگر اندازه‌گیری می‌کند. مقدار این ضریب بین -۱ و ۱ متغیر است؛ هرچه این مقدار به ۱ نزدیک‌تر باشد، خوشه‌بندی بهتر است. مقدار ضریب سیلوئت محاسبه شده چاپ می‌شود تا کیفیت خوشه‌بندی ارزیابی شود.




شکل ۳: ارزیابی با معیار سیلوئت

### ۱-۱-۱-بهینه‌سازی هایپرپارامترهای AGNES

حال به بهینه‌سازی هایپرپارامتر linkage می‌پردازیم. برای این منظور، ابتدا یک تابع می‌نویسیم که از کلاس AgglomerativeClustering یک شی با هایپرپارامتر مورد نظر ما می‌سازد. این هایپرپارامتر به عنوان آرگومان وارد تابع می‌شود. سپس خوشه‌بندی انجام می‌شود و معیار سیلوئت مشابه بخش قبلی محاسبه می‌شود. سپس این مقدار بازگردانده می‌شود. شکل (۴) این تابع را نمایش می‌دهد.




شکل ۴: تابع محاسبه معیار سیلوئت با هایپرپارامترهای مختلف

حال در یک لیست، مقادیری که این هایپرپارامتر می‌تواند دریافت کند را قرار می‌دهیم و یک حد پایین و یک متغیر برای نگهداری بهترین نتیجه ایجاد می‌کنیم. سپس در یک حلقه، به ازای تمامی مقادیر هایپرپارامتر، تابع ایجاد شده را اجرا می‌کنیم. سپس در بلوک بعددی، بهترین نتیجه و هایپرپارامتر متناظر با آن را نمایش می‌دهیم. شکل (۵) این دو بلوک و خروجی آنها را نمایش می‌دهد.

حال یک بار دیگر خوشه‌بندی را با هایپرپارامتر بهینه انجام می‌دهیم. نحوه انجام خوشه‌بندی مشابه قبل است. سپس سه معیار میانگین فاصله بین اعضای یک خوشه، حداقل فاصله بین خوشه‌ها، و مقدار سیلوئت را محاسبه می‌کنیم.

برای محاسبه میانگین فاصله بین اعضای هر خوشه از یک تابع استفاده می‌کنیم ابتدا برچسب‌های منحصر به فرد خوشه‌ها شناسایی می‌شوند. سپس تابع برای هر خوشه، فاصله‌های جفتی بین نقاط محاسبه می‌کند و میانگین این فاصله‌ها ذخیره می‌گردد. در نهایت، میانگین کلی این فاصله‌ها به عنوان یک معیار کلی بازگشت داده می‌شود. این میانگین فاصله درون خوشه‌ای برای ارزیابی کیفیت خوشه‌بندی استفاده می‌شود؛ مقادیر پایین‌تر نشان‌دهنده خوشه‌های متراکم‌تر و کیفیت بهتر خوشه‌بندی است. شکل (۶) این تابع و خروجی آن را نمایش می‌دهد.

Searching for the best hyper-parameter

```
linkages = ["ward", "complete", "average", "single"]
best_linkage = None
best_score = -1

for linkage in linkages:
    try:
        score = compute_silhouette_score(X, linkage)
        print(f"Silhouette score for linkage={linkage}: {score}")
        if score > best_score:
            best_score = score
            best_linkage = linkage
    except Exception as e:
        print("Failed")
```

✓ 0.0s

```
Silhouette score for linkage=ward: 0.5479462758760916
Silhouette score for linkage=complete: 0.6166354826398824
Silhouette score for linkage=average: 0.5986986692237101
Silhouette score for linkage=single: 0.3742447061332927
```

```
print(f"Best linkage: {best_linkage}")
print(f"Best silhouette score: {best_score}")
```

✓ 0.0s

```
Best linkage: complete
Best silhouette score: 0.6166354826398824
```

شکل ۵: پیدا کردن هایپرپارامتر بهینه برای اگنس

```
# Function to calculate the average within-cluster distance
def average_within_cluster_distance(X, labels):
    unique_labels = np.unique(labels)
    average_distances = []

    for label in unique_labels:
        cluster_points = X[labels == label]
        # Calculate pairwise distances within the cluster
        if len(cluster_points) > 1:  # Ensure there are at least two points to calculate distance
            distances = pairwise_distances(cluster_points)
            avg_distance = np.sum(distances) / (2 * len(cluster_points))
            average_distances.append(avg_distance)
        else:
            average_distances.append(0)  # If only one point, distance is zero

    # Calculate the average of the averages for a global measure
    overall_average = np.mean(average_distances)
    return overall_average

# Calculate the average within-cluster distance
avg_distance = average_within_cluster_distance(X, labels)
print("Average Within-Cluster Distance:", avg_distance)
```

✓ 0.0s

```
Average Within-Cluster Distance: 13064.78367175092
```

شکل ۶:محاسبه میانگین فاصله اعضای هر خوشه

برای محاسبه حداقل فاصله بین خوشه‌ها مجدداً دو تابع می‌نویسیم. تابع اول مراکز هر خوشه را محاسبه می‌کند. این مراکز به عنوان میانگین مختصات نقاط درون هر خوشه تعیین می‌شوند و نتیجه به صورت یک آرایه از مراکز خوشه‌ها بازگشت داده می‌شود. سپس یک تابع برای محاسبه

فاصله‌های جفتی بین مراکز خوشه‌ها ایجاد می‌شود. فاصله‌های محاسبه شده در یک ماتریس ذخیره می‌شوند و مقدار روی قطر اصلی ماتریس (فاصله مرکز خوشه به خود) با مقدار بی‌نهایت پر می‌شود تا این فاصله‌ها در محاسبه حداقل فاصله تأثیری نداشته باشند. سپس حداقل فاصله بین دو مرکز خوشه مختلف استخراج می‌شود. شکل (۷) این دو تابع و مقاد خروجی آنها را نمایش می‌دهد.

2-Between-Cluster Distance

```
# Calculate cluster centroids
def calculate_centroids(X, labels):
    unique_labels = np.unique(labels)
    centroids = []
    for label in unique_labels:
        centroids.append(np.mean(X[labels == label], axis=0))
    return np.array(centroids)

# Calculate centroids
centroids = calculate_centroids(X, labels)

# Calculate the minimum distance between any two centroids
def min_inter_cluster_distance(centroids):
    dist_matrix = pairwise_distances(centroids)
    np.fill_diagonal(dist_matrix, np.inf)  # Fill diagonal with infinite to ignore zero distance of clusters to themselves
    return np.min(dist_matrix)

# Get the minimum inter-cluster distance
min_distance = min_inter_cluster_distance(centroids)
print("Minimum Between-Cluster Distance:", min_distance)
```

✓ 0.0s

```
Minimum Between-Cluster Distance: 278.0000296564606
```

شکل ۷: محاسبه حداقل فاصله بین خوشه‌ها

سپس معیار سیلوئت محاسبه می‌شود که برابر با همان مقدار قبلی است.

### ۱-۲-خوشه‌بندی کا میانگین

کد شکل (۸) برای خوشه‌بندی داده‌ها با استفاده از مدل K-Means طراحی شده است. ابتدا یک نمونه از مدل K-Means با تنظیمات مشخص ایجاد می‌شود که تعداد خوشه‌ها، روش شروع k-means++، حداکثر تعداد تکرارها، تعداد شروع‌های مختلف برای انتخاب بهترین نتیجه، و تنظیمات حالت تصادفی برای اطمینان از تکرارپذیری نتایج را شامل می‌شود. سپس مدل با داده‌های ویژگی‌ها برازش داده می‌شود که داده‌ها را به دو خوشه تقسیم می‌کند.

```
# Instantiate the model
clustering = KMeans(n_clusters=2, init='k-means++', max_iter=300, n_init=10, random_state=0)
# Fit model
clustering.fit(X)
```

✓ 0.0s

```
c:\Users\Lenovo\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1446: UserWarning: KMeans
  warnings.warn(
```

```
KMeans(n_clusters=2, n_init=10, random_state=0)
```

شکل ۸: خوشه‌بندی اولیه با کا-میانگین

سپس مانند بخش قبلی معیار سیلوئت این خوشه‌بندی را محاسبه می‌کنیم. شکل (۹) مقدار این معیار را نمایش می‌دهد.

```
Silhouette Coefficient:  0.6253209536141464
```

شکل ۹: مقدار معیار سیلوئت برای خوشه‌بندی کا میانگین اولیه

حال به بهینه‌سازی هایپرپارامترهای این مدل می‌پردازیم.

### ۱-۲-۱-بهینه‌سازی هایپرپارامترها

برای انجام بهینه سازی، مجدداً همانند قبل یک تابع می‌نویسیم که پارامترها را به عنوان آرگومان دریافت می‌کند، خوشه‌بندی را انجام می‌دهد و معیار سیلوئت را محاسبه می‌کند. شکل (۱۰) این تابع را نمایش می‌دهد.

```python
def compute_silhouette_score(X, n_init, max_iter, tol):
    clustering = KMeans(n_clusters=2, n_init=n_init, max_iter=max_iter, tol=tol, random_state=0)
    cluster_labels = clustering.fit_predict(X)
    score = silhouette_score(X, cluster_labels)
    return score
```

✓ 0.0s

شکل ۱۰: تابع مورد استفاده برای بهینه‌سازی هایپرپارامترهای کا میانگین

سپس همانند بخش قبلی برای هر یک از سه هایپرپارامتر هدف، یک لیست می‌نویسیم. سپس برای همه حالات ممکن، تابع بالا را فراخوانی می‌کنیم و نتایج را ذخیره می‌کنیم. شکل (۱۱) بلوکی که بهینه‌سازی هایپرپارامترها انجام می‌دهد نمایش می‌دهد.

```python
n_init_values = [10, 20, 30]
max_iter_values = [300, 400, 500]
tol_values = [1e-4, 1e-3, 1e-2]

best_params = None
best_score = -1

for n_init in n_init_values:
    for max_iter in max_iter_values:
        for tol in tol_values:
            score = compute_silhouette_score(X, n_init, max_iter, tol)
            print(f"Silhouette score for n_init={n_init}, max_iter={max_iter}, tol={tol}: {score}")
            if score > best_score:
                best_score = score
                best_params = (n_init, max_iter, tol)
```

✓ 1.3s

شکل ۱۱: بهینه‌سازی هایپرپارامترهای کا-میانگین

شکل (۱۲) نتایج حاصل از بهینه‌سازی را نمایش می‌دهد.

```
print(f"Best parameters: n_init={best_params[0]}, max_iter={best_params[1]}, tol={best_params[2]}")
print(f"Best silhouette score: {best_score}")
```
✓ 0.0s

```
Best parameters: n_init=10, max_iter=300, tol=0.0001
Best silhouette score: 0.6253164998557754
```

شکل ۱۲: نتیجه بهینه‌سازی هایپرپارامترهای کا-میانگین

حال مجدداً مشابه بخش قبلی مقدار میانگین فاصله بین اعضای هر خوشه را محاسبه می‌کنیم. شکل (۱۳) این مقدار را نمایش می‌دهد.

```
Average Within-Cluster Distance: 12883.301690276458
```

شکل ۱۳: میانگین فاصله بین اعضای هر خوشه در کا-میانگین

سپس مقدار Compactness را با استفاده از اتریبیوت _inertia. نمایش می‌دهیم. شکل (۱۴) این مقدار را نمایش می‌دهد.

```
clustering.inertia_
```
✓ 0.0s

```
3581110.8163670027
```

شکل ۱۴: مقدار Compactness

سپس همانند بخش قبلی، حداقل فاصله بین خوشه‌ها را محاسبه می‌کنیم.

```
Minimum Between-Cluster Distance: 278.00007876369637
```

شکل ۱۵: حداقل فاصله بین خوشه‌ها در کا-میانگین

و در نهایت مشابه بخش‌های قبلی معیار سیلوئت را محاسبه می‌کنیم که مقدار آن در شکل (۱۶) نمایش داده شده است.

```
Silhouette Coefficient:  0.6253209536141464
```

شکل ۱۶: محاسبه معیار سیلوئت برای کا-میانگین

## ۲- پاسخ سوال دوم

داده ها را با روش های«شبکه پرسپترون چند لایه» دسته بندی کنید. روش خود برای یافتن ساختار مناسب شبکه را بیان کنید از روش اعتبارسنجی سه بخشی استفاده کنید و نتایج را گزارش کنید.

**۲-۱- شبکه عصبی پرسپترون چند لایه**

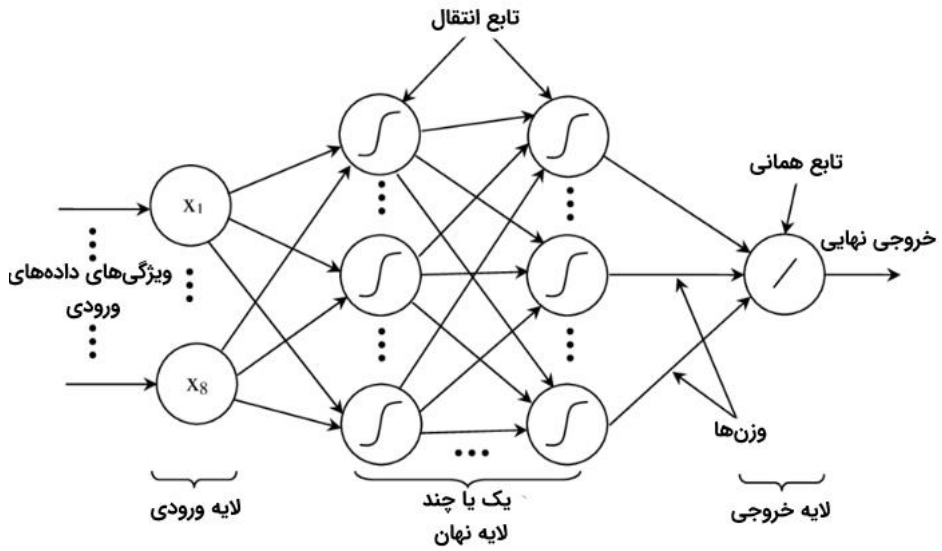
شبکه عصبی پرسپترون چند لایه از چندین لایه متصل به هم تشکیل شده است. در ادامه، به لایه‌های این مدل اشاره می‌کنیم:

- ❖ لایه ورودی: این لایه، اولین لایه شبکه عصبی است که داده‌های ورودی را دریافت می‌کند و وظیفه آن، ارسال داده‌ها به لایه بعدی است.
- ❖ لایه پنهان: لایه‌های پنهان می‌توانند بیش از یک لایه باشند. این لایه‌ها از نورون‌هایی تشکیل شده‌اند که وظیفه انجام محاسباتی را بر روی ورودی‌های خود بر عهده دارند و خروجی حاصل شده از محاسبات را به لایه بعد خود منتقل می‌کنند.
- ❖ لایه خروجی: این لایه، مقدار خروجی نهایی شبکه را مشخص می‌کند. ورودی این لایه، خروجی‌های لایه پنهان است.

۲-۱-۱- پردازش داده ها در شبکه عصبی پرسپترون چند لایه

همان‌طور که در بخش پیشین گفته شد، مدل پرسپترون چند لایه از سه لایه اصلی ورودی، لایه پنهان و لایه خروجی تشکیل شده است. زمانی که داده‌ها را به ورودی این شبکه ارسال می‌کنیم، داده‌ها از لایه ورودی عبور می‌کنند. در این لایه، هیچ‌گونه محاسباتی بر روی داده‌ها انجام نمی‌شود و لایه ورودی صرفاً مسئولیت انتقال داده‌ها را به لایه بعدی بر عهده دارد. نورون‌های لایه پنهان، بر روی داده‌های دریافتی محاسباتی را انجام می‌دهند و سپس مقادیر حاصل شده را به لایه بعدی خود منتقل می‌کنند. این روند انتقال داده‌ها تا زمانی پیش می‌رود که داده‌ها به لایه آخر منتقل شوند.

نورون‌های لایه‌های پنهان با یک سری مقادیر (وزن‌های شبکه) به یکدیگر متصل هستند که میزان اهمیت نورون را برای پردازش داده‌ها مشخص می‌کند. هر چقدر وزن‌ها بیشتر باشند، تاثیر نورون متصل به آن وزن در محاسبه بیشتر است و در پی این اتفاق، آن نورون بر روی محاسبه خروجی مسئله تاثیر بیشتری خواهد گذاشت. از طرف دیگر، اگر وزن یک نورون خیلی کوچک باشد، تاثیر داده آن نورون در محاسبات بعدی شبکه کم‌تر خواهد بود و در نتیجه تغییرات چندانی در خروجی نهایی شبکه ایجاد نخواهد کرد.




شکل ۱۷: ساختار درونی مدل پرسپترون چند لایه

**۲-۲- تقسیم‌بندی داده‌ها به آموزشی و آزمایشی**

در این قسمت داده‌ها را به داده‌های آموزشی و آزمایشی تقسیم کردیم که لازم به ذکر است که دیتا ست استفاده شده خروجی پردازش داده‌ها در تمرین ۱ است که همچنین با استفاده از الگوریتم ژنتیک تعداد ویژگی برای آن مشخص شده است با توجه به اینکه بهترین الگورتیم از نظر میزان دقت بوده است.

```
# We have to balance our training set

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=0, stratify = Y)
```

شکل ۱۸: تقسیم‌بندی داده‌ها به آموزشی و آزمایشی

با مشاهده نتیجه دریافتیم که نیاز است کلاس‌ها را به تعادل برسانیم.

### ۲-۲-۱- متعادل کردن تقسیم بندی داده‌های آموزشی و آزمایشی

متعادل کردن داده‌ها یکی از مهمترین مراحل پیش پردازش است. هدف اصلی متعادل کردن داده ها، افزایش دقت مدل‌های پیش‌بینی است. عدم توازن داده‌ها به دلیل تفاوت توزیع کلاس‌ها در داده‌های آموزشی، ممکن است موجب ایجاد مدل هایی با دقت پایین یا حتی یک مدل پیش‌بینی کننده از همه یک کلاس بیشتر شود. پیش از آن‌که به تمرین خود بپردازیم بهتر است روش‌های متعادل کردن داده‌ها را بررسی کنیم.

- روش Over-Sampling

داده‌های را ایجاد می‌کند، به عبارت دیگر عملیات تکثیر داده‌های کلاس‌های کم را انجام می‌دهد. در واقع زمانی است که فاصله بین دوتا کلاس خیلی زیاد باشد و به همین دلیل لازم باشد با افزایش یک کلاس، به کلاس بعدی حالت تعادل را به دست آوریم. این افزایش تعداد نمونه‌های هر کلاس فرصت بیشتری برای آموزش به مدل می‌دهد. کدهای اجرا شده برای این روش به شرح زیر می‌باشد.

```
# Over Sampling

sm = SMOTE(random_state = 2)

print("\nClass 1 before Over Sampling --> ", sum(Y_train == 1))
print("\nClass 0 before Over Sampling --> ", sum(Y_train == 0))


# X_train after Over Sampling --> X_train_OS
# Y_train after Over Sampling --> Y_train_OS
X_train_OS, Y_train_OS = sm.fit_resample(X_train, Y_train.ravel())

print("\nThe shape of the X after Over Sampling -->", X_train_OS.shape)
print("\nThe shape of the Y after Over Sampling -->", Y_train_OS.shape)

print("\nClass 1 after Over Sampling --> ", sum(Y_train_OS == 1))
print("\nClass 0 after Over Sampling --> ", sum(Y_train_OS == 0))
print("\n")
```

```
Class 1 before Over Sampling -->  127

Class 0 before Over Sampling -->  206

The shape of the X after Over Sampling --> (412, 15)

The shape of the Y after Over Sampling --> (412,)

Class 1 after Over Sampling -->  206

Class 0 after Over Sampling -->  206
```

شکل ۱۹: کد و خروجی اجرا شده برای متعادل کردن تقسیم‌بندی داده‌های آموزشی

### ۲-۳- ساخت مدل پایتورچ

در این قسمت برای شروع کار و ساخت شبکه، مدل پایتورچ را میسازیم و در قسمت بعد داده‌های به تعادل رسیده در قسمت آموزش و تست را از فرمت دیتا فریم به فرمت تنسور انتقال می‌دهیم که داده های این بخش در شکل زیر قرار گرفته اند.

## B. Building PyTorch model

```
# Standard PyTorch imports
import torch
from torch import nn

# Make device agnostic code
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

```
'cpu'
```

```
# Dataframe to tensor and device
X_train = torch.tensor(X_train_OS.values, dtype = torch.float32).to(device)
y_train = torch.tensor(Y_train_OS, dtype = torch.float32).to(device)
X_test = torch.tensor(X_test.values, dtype = torch.float32).to(device)
y_test = torch.tensor(Y_test.values, dtype = torch.float32).to(device)
```

شکل ۲۰: ساخت مدل پایتورچ و انتقال به تنسور

**۲-۴- ساخت مدل های اولیه شبکه**

برای بررسی مدل و در نهایت تنظیم کردن پارامترهای مختلف ابتدا مدل را با تعداد لایه‌های پنهان متفاوت آموزش دادیم و میزان دقت را بررسی کردیم.

در مدل اول ابتدا یک شبکه با یک لایه پنهان با ۸ نرون را ساخته شد که در نهایت با پارامترهای دیگر مشخص شده به صورت ثابت دقت به دست آمده حدود ۸۳ درصد به دست آمد. کد و خروجی مدل شبکه اول در شکل زیر قرار داده است:

## B.1 Model 0

1 hidden layer with 8 neurons

### Constructing the model

```
param_grid_mlp = {
    'hidden_layer_sizes': [(8)],
    'activation': ['relu'],
    'solver': ['sgd'],
    'early_stopping': [True],
    'learning_rate': ['constant', 'invscaling', 'adaptive'],
    'learning_rate_init' : [0.01],
    'max_iter': [1000],
    'random_state': [42]}
CV = StratifiedKFold(n_splits = 3)
model_0 = GridSearchCV(MLPClassifier(), param_grid_mlp, cv=CV,
                       scoring='accuracy',verbose=False)
model_0.fit(X_train, y_train)
```

```
GridSearchCV
estimator: MLPClassifier
MLPClassifier
```

```
model_0.best_score_
```

```
0.8299481646038295
```

شکل ۲۱: کد و خروجی مدل شبکه ۱

سپس برای تست اینکه آیا افزایش تعداد لایه‌های پنهان شبکه بهبودی در شبکه ایجاد می‌کند یا خیر مدل دیگری این بار با دو لایه پنهان که هر کدام از لایه‌ها ۸ نرون در آن وجود دارد ساخته شد که در نهایت با افزایش تعداد لایه پنهان در شبکه دقت آن به حدود ۹۳ درصد رسید. کد و خروجی مدل شبکه دوم در شکل زیر قرار داده است:

## B.2 Model 1

2 hidden layer with 8 neurons

### Constructing the model

```
param_grid_mlp = {
    'hidden_layer_sizes': [(8,8)],
    'activation': ['relu'],
    'solver': ['sgd'],
    'early_stopping': [True],
    'learning_rate': ['constant', 'invscaling', 'adaptive'],
    'learning_rate_init' : [0.01],
    'max_iter': [1000],
    'random_state': [42]}
CV = StratifiedKFold(n_splits = 3)
model_1 = GridSearchCV(MLPClassifier(), param_grid_mlp, cv=CV,
                       scoring='accuracy',verbose=False)
model_1.fit(X_train, y_train)
```

GridSearchCV
estimator: MLPClassifier
MLPClassifier

```
model_1.best_score_
```

```
0.9320674212771959
```

شکل ۲۲: کد و خروجی مدل شبکه ۲

**۲-۵- تنظیم پارامتر های شبکه و اجرا بهترین مدل**

در این بخش به شرح شکل ۲۳ همه پارامترهایی که می‌خواهیم تنظیم کنیم و مقادیر که مد نظر داریم را در کد ذکر کرده ایم. به صورت کلی پارامتر اول ما تعداد لایه پنهان و نورون ما بوده است که مقدار (۸) یعنی ۱ لایه پنهان داریم با ۸ نورون یا مقدار (۸،۱۶) یعنی دوتا لایه پنهان داریم که اولی ۸ نورون و دومی ۱۶ نورون دارد و مقدار (۸،۱۶،۸) یعنی سه تا لایه پنهان داریم که اولی ۸ نورون و دومی ۱۶ و سومی ۸ نورون دارد.

- Activation لیستی از رشته هاست که تابع فعال سازی را برای لایه های پنهان مشخص می کند. مقادیر ممکن tanh و relu هستند.
- solver لیستی از رشته هایی است که حل کننده را برای بهینه سازی وزن مشخص می‌کند. مقادیر ممکن adam و sgd هستند.
- early_stoppin فهرستی از بولی‌ها است که مشخص می‌کند در زمانی که امتیاز اعتبارسنجی بهبود نمی‌یابد از توقف زودهنگام برای پایان دادن به آموزش استفاده شود یا خیر. مقدار در این مورد True است.
- Learn_rate لیستی از رشته‌هایی است که زمان‌بندی نرخ یادگیری را برای به‌روزرسانی وزن مشخص می‌کند. مقادیر ممکن ثابت، invscaling و تطبیقی هستند.
- Learning_rate_init لیستی از شناورها است که نرخ یادگیری اولیه استفاده شده را مشخص می‌کند. مقادیر ممکن ۰.۱، ۰.۰۱، ۰.۰۰۱ و ۰.۰۵ هستند.
- max_iter لیستی از اعداد صحیح است که حداکثر تعداد تکرارها را برای حل کننده مشخص می‌کند. مقادیر در این مورد ۱۰۰۰و ۲۰۰۰ است.
- random_state لیستی از اعداد صحیح است که دانه تصادفی را برای تکرارپذیری مشخص می کند. مقدار در این مورد ۴۲ است.

- CV متغیری است که نمونه‌ای از StratifiedKFold را ذخیره می‌کند، که یک تقسیم‌کننده اعتبارسنجی متقاطع است که درصد نمونه‌ها را برای هر کلاس حفظ می‌کند. پارامتر n_splits روی ۳ تنظیم شده است، به این معنی که داده ها به ۳ برابر تقسیم می‌شوند.

model_۲ متغیری است که نمونه ای از GridSearchCV را ذخیره می‌کند، که کلاسی است که جستجوی شبکه را روی شبکه پارامتر انجام می دهد. پارامترها به شرح زیر است:

- estimator شی برآوردگر است که باید بهینه شود، که در این مورد MLPClassifier است.
- param_grid شبکه پارامتری است که باید جستجو شود که در این مورد param_grid_mlp است.
- cv استراتژی اعتبارسنجی متقاطع مورد استفاده است که در این مورد CV است.
- امتیازدهی معیار عملکردی است که برای ارزیابی نامزدها استفاده می شود که در این مورد دقت است.
- verbose سطح پرحرفی است که در این مورد False است.

لازم به ذکر است که MLPClassifier از Cross-Entropy به عنوان loss function به طور پیش فرض پشتیبانی می کند

model_۲.fit (X_train, y_train). روشی است که با استفاده از بهترین پارامترهای یافت شده (الگوریتم GA) که در این مورد X_train و y_trainهستند، برآوردگر را برازش می‌کند.

پس از تنظیم پارامترها، بهترین پارامترهای ممکن به ما داده می‌شود که مانند قبل از آن استفاده می‌کنیم.

## C.Tuning

```
param_grid_mlp = {
    'hidden_layer_sizes': [(8),(16), (32), (8,4),(8,8),(16,16),(16,8),(32,32),(8,4,8),(8,16,8),(8,8,8)],
    'activation': ['tanh', 'relu'],
    'solver': ['adam', 'sgd'],
    'early_stopping': [True],
    'learning_rate': ['constant', 'invscaling', 'adaptive'],
    'learning_rate_init' : [0.1, 0.01, 0.05, 0.001],
    'max_iter': [1000,2000],
    'random_state': [42]}
CV = StratifiedKFold(n_splits = 3)
model_2 = GridSearchCV(MLPClassifier(), param_grid_mlp, cv=CV,
                       scoring='accuracy',verbose=False)
model_2.fit(X_train, y_train)
```

GridSearchCV
estimator: MLPClassifier
MLPClassifier

```
model_2.best_params_
```

```
{'activation': 'relu',
 'early_stopping': True,
 'hidden_layer_sizes': (8, 16, 8),
 'learning_rate': 'constant',
 'learning_rate_init': 0.01,
 'max_iter': 1000,
 'random_state': 42,
 'solver': 'adam'}
```

شکل ۲۳: تنظیم پارامترها برای MLP

که دقت بدست آمده برای داده های آموزشی با استفاده از بهترین پارامتر ها به شرح شکل زیر است:

```
1  model_2.best_score_
```

```
0.9708910751436933
```

شکل ۲۴: دقت روی داده آموزشی بعد تنظیم پارامتر

سپس شبکه به دست آمده را که شامل بهترین پارامترهای تنظیم شده است روی دیتاها و آموزش داده و مدل شبکه عصبی به دست آمده با توجه به اینکه با روش اعتبارسنجی سه بخشی انجام شده است برای به دست آوردن دقت‌های هر کدام از بخش‌های مختلف این سه مرحله کد زیر روی شبکه اجرا شد که در نهایت دقت به دست آمده میانگینی از دقت به دست آمده در هر کدام از بخش‌های مختلف اعتبارسنجی است.

```
1  model_3 = MLPClassifier(**model_2.best_params_)
```

```
1  model_3.fit(X_train, y_train)
2  CV = StratifiedKFold(n_splits = 3)
3  CV_scores = cross_val_score(model_3, X_train, y_train, cv=CV)
4  CV_score_df = pd.DataFrame(CV_scores, columns = ['Accuracy'])
5  display(CV_score_df)
6  print('The Average of CV 3 Fold Scores -->', CV_score_df.Accuracy.mean())
```

| | Accuracy |
|---|---|
| 0 | 0.963768 |
| 1 | 0.978102 |
| 2 | 0.970803 |

```
The Average of CV 3 Fold Scores --> 0.9708910751436933
```

شکل ۲۵: آموزش شبکه با بهترین پارامتر ها با اعتبارسنجی سه بخشی

**۲-۶- اجرا مدل با پارامترهای مناسب بر روی داده تست و بررسی شاخص‌ها**

حال بهترین مدل به دست آمده را روی داده‌های تست بررسی می‌کنیم و با استفاده از دو مقادیر لیبل‌های داده‌های تست و همچنین لیبل‌های پیش‌بینی شده توسط مدل با شاخص‌های ماتریس پراکندگی مدل را بررسی می‌کنیم.

## Implementation on test data

```python
y_predict = model_3.predict(X_test)
confusion = confusion_matrix(Y_test, y_predict)
accuracy = accuracy_score(Y_test, y_predict)
precision = precision_score(Y_test, y_predict)
recall = recall_score(Y_test, y_predict)
f1 = f1_score(Y_test, y_predict)
auc = roc_auc_score(Y_test, y_predict)

# Print the results
print("Confusion matrix:")
print(confusion)
print('\nReport:\n', classification_report(Y_test, y_predict))
print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Precision: {:.2f}%".format(precision * 100))
print("Recall: {:.2f}%".format(recall * 100))
print("F1 score: {:.2f}%".format(f1 * 100))
print("AUC: {:.2f}%".format(auc * 100))
```

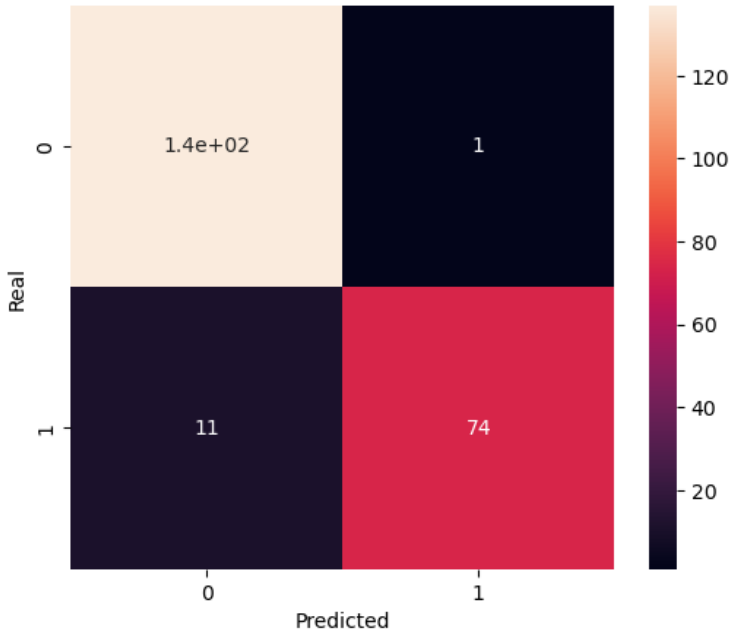
شکل ۲۶: کد مربوط به اجرای شبکه بر روی داده های تست

```
Confusion matrix:
[[137   1]
 [ 11  74]]

Report:
              precision    recall  f1-score   support

         0.0       0.93      0.99      0.96       138
         1.0       0.99      0.87      0.93        85

    accuracy                           0.95       223
   macro avg       0.96      0.93      0.94       223
weighted avg       0.95      0.95      0.95       223

Accuracy: 94.62%
Precision: 98.67%
Recall: 87.06%
F1 score: 92.50%
AUC: 93.17%
```

شکل ۲۷: خروجی شاخص های مختلف با اجرای رو داده تست

همچنین می‌توان فهمید با توجه به اختلاف کم بین شاخص‌های مختلف در ماتریس درهم ریختگی شبکه ما بر روی داده‌های تست به خوبی عمل کرده است.

همچنین برای دید بهتر ماتریس درهم‌ریختگی می‌توانیم در شکل زیر نمای بهتری از آن داشته باشیم.




شکل ۲۸: نمودار ماتریس درهم‌ریختگی

## ۳-پاسخ سوال سوم

ج - حال داده‌ها را با شبکه عصبی با تابع محرک شعاعی دسته‌بندی کنید پارامترهای توابع محرک شعاعی تعداد نرون‌ها مراکز و پهنای توابع شعاعی را با روش خوشه‌بندی کامیانگین بدست آورید. ۶۰ درصد داده‌ها را برای آموزش و الباقی را برای آزمایش به کار ببرید ماتریس در هم ریختگی را تشکیل داده و نتایج مندرج در آن را بر اساس شاخص‌های مختلف شرح دهید.

با توجه به خواسته سوال، ابتدا داده‌ها را به دو دسته آموزش و آزمایش تقسیم می‌کنیم. ۶۰ درصد برای آموزش و ۴۰ درصد باقیمانده برای آموزش استفاده می‌شوند. کد مربوط به شکل ۲۹ نمایانگر این خواسته می‌باشد.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=4)
```

شکل ۲۹: تقسیم داده‌ها به دو کلاس آموزشی و آزمایشی

### ۳-۲ روش K-Means

الگوریتم K-Means یک الگوریتم یادگیری بدون نظارت است که برای حل مشکلات خوشه‌بندی در یادگیری ماشین یا علم داده استفاده می‌شود. خوشه‌بندی K-Means روشی در کمی‌سازی بردارهاست که در اصل از پردازش سیگنال گرفته شده و برای آنالیز خوشه‌بندی در داده‌کاوی محبوب است. هدف الگوریتم K-Means خوشه‌بندی k مشاهده به n خوشه است که در آن هریک از مشاهدات متعلق به خوشه‌ای با نزدیکترین میانگین به آن است، این میانگین به عنوان نمونه استفاده می‌شود.

در شکل ۳۰ کد مربوط به الگوریتم K-Means را مشاهده می‌کنید. این الگوریتم داده‌ها بر اساس n برابر ۲ را خوشه‌بندی می‌کند و طبق n_init = ۱۲ ، دوازده بار مرکز خوشه را محاسبه کرده است.

```
k_means = KMeans(n_clusters=2, init='k-means++', n_init=12)
```

```
k_means.fit(X_train_OS)
```

```
KMeans
KMeans(n_clusters=2, n_init=12)
```

شکل ۳۰: کد الگوریتم K-Means

برچسب‌های تولید شده در این روش را در شکل ۳۱ می‌توانید مشاهده کنید. مراکز خوشه‌ها نیز در شکل ۳۲، قابل مشاهده است.

```
k_means_labels=k_means.labels_
k_means_labels
```

```
array([0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1])
```

شکل ۳۱: برچسب‌های تولید شده از روش K-Means

```
k_means_cluster_centers= k_means.cluster_centers_
k_means_cluster_centers
```

```
array([[0.27711926, 0.28360241, 0.2054537 , 0.4057418 , 0.1640564 ,
        0.39778269, 0.37464976, 0.16726199, 0.36876769, 0.16217431,
        0.34894233, 0.39592396, 0.15606799, 0.37401827, 0.23259601,
        0.25380919, 0.35565212],
       [0.55953043, 0.5814711 , 0.49750104, 0.56118571, 0.53445277,
        0.56874718, 0.42569463, 0.41691638, 0.36695005, 0.40133235,
        0.33357755, 0.5588771 , 0.46339273, 0.53265123, 0.48625087,
        0.63179293, 0.53048666]])
```

شکل ۳۲: مراکز تولید شده توسط روش K-Means

### ۳-۳ روش RBF[1]

شبکه‌های عصبی RBF گونه‌ای خاص از شبکه‌های عصبی مصنوعی به حساب می‌آیند که مبتنی بر فاصله‌اند و شباهت بین داده‌ها را براساس فاصله می‌سنجند. یک شبکه RBF نوعی از شبکه عصبی مصنوعی شبکه عصبی روبه جلو (Feed Forward) است که از سه لایه تشکیل می‌شود. هر یک از این لایه در ادامه فهرست شده‌اند:

۱. لایه ورودی
۲. لایه پنهان
۳. لایه خروجی

حال با استفاده از برچسب‌ها و مراکز تولید شده در بخش قبل، الگوریتم RBF را اجرا می‌کنیم. کد اجرای این الگوریتم، در شکل ۳۳و خروجی آن در شکل ۳۴ قابل مشاهده است.

```
# Perform K-Means clustering
cluster_centers =k_means_cluster_centers
k_means_labels = k_means_labels

# Use the RBF kernel to create a classification model
kernel = 1.0 * RBF(1.0)
model = GaussianProcessClassifier(kernel=kernel)
model.fit(X_train_OS, k_means_labels)

# Predict labels for the test set
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Calculate confusion matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", confusion_matrix)
```

شکل ۳۳: کد پیاده‌سازی الگوریتم RBF

```
Accuracy: 0.9506726457399103
Confusion Matrix:
 [[137   1]
 [ 10  75]]
```

شکل ۳۴: کد خروجی الگوریتم RBF

[1] Radial basis function

با توجه به نتایج به دست آمده در شکل ۳۵، مشاهده می‌کنیم که الگوریتم RBF، با استفاده از برچسب‌ها و مراکز تولید شده با الگوریتم K-Means، عملکرد خوبی را برای مجموعه داده مورد بررسی ما داشته است و پیش‌بینی‌ها با دقت ۹۵ درصد، خروجی مطلوبی را نمایش می‌دهد.

در شکل‌های ۳۵، ۳۶، ۳۷ و ۳۸ سایر شاخص‌های ارزیابی را بررسی کرده‌ایم.

## Sensitivity ¶

```
: print(' Sensitivity :', recall_score(y_test, y_pred))
```

```
Sensitivity : 0.8823529411764706
```

شکل ۳۵: ارزیابی معیار حساسیت

## precision, recall, f1-score,support

```
print (classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.99      0.96       138
         1.0       0.99      0.88      0.93        85

    accuracy                           0.95       223
   macro avg       0.96      0.94      0.95       223
weighted avg       0.95      0.95      0.95       223
```

شکل ۳۶: ارزیابی سایر معیارها

## f2_score

```
f2_score = fbeta_score(y_test, y_pred, beta=2, average='macro')

print("F2-score:", f2_score)
```

```
F2-score: 0.9407068476945086
```

شکل ۳۷: ارزیابی معیار f۲score-

## MCC ¶

```
mcc = matthews_corrcoef(y_test,  y_pred)

print("Matthews Correlation Coefficient:", mcc)
```

```
Matthews Correlation Coefficient: 0.8966944549327335
```

شکل ۳۸: ارزیابی معیار mcc

با ارزیابی تمامی معیارهای نمایش داده شده، و به دست آوردن اعدادی بیشتر از ۸۸ درصد ، میتوان اطمینان حاصل کرد که مدل RBF ، عملکردی دقیق و مطلوب داشته است.