دانشگاه تهران

دانشکده علوم مهندسی

یادگیری ماشین

تمرین پنجم

دکتر سایه میرزایی

سپیده فاطمی خوراسگانی

شماره دانشجویی: 810897059

بهار 00

سؤال ۱)

۱) روش K_means و روابط آن:

الگوریتم K_means یک روش خوشه بندی در مسائل unsupervised learning (یادگیری غیر نظارتی) است. در این مسائل ما برچسب‌های داده‌ها یا همان y ها را نداریم.

در این روش ما تلاش می‌کنیم که داده‌های مشابه هم را در یک دسته قرار دهیم. معیار این شباهت بستگی به الگوریتمی که استفاده می‌کنیم دارد. در k_means این معیار نزدیکی نقاط به یکدیگر می باشد.

در این روش هر داده فقط در یکی از k دسته قرار می‌گیرد و درواقع دسته ها با هم همپوشانی ندارند.

فرضیات: در این الگوریتم k خوشه و m داده داریم که هر کدام اندازه n دارند.

الگوریتم k means:

۱) ابتدا تعداد خوشه ها k را مشخص می کنیم.

۲) نقاط centroid را به صورت رندوم و از بین داده‌ها انتخاب می کنیم. (k داده از m داده را انتخاب می‌کنیم که می‌دانیم حتماً k<m است)

۳) حلقه زیر را تا زمانی که دیگر نقاط centroid تغییری نکنند ادامه می دهیم.

۴) برای هر داده مقدار فاصله آن نقطه از هر centroid را محاسبه می‌کنیم . کمترین مقدار محاسبه شده همان خوشه مربوط به آن داده می باشد.

۵) نقاط centroid را update می‌کنیم. به این صورت که هر centroid برابر با میانگین تمام داده‌هایی که در آن خوشه قرار دارند می شود.

و یا به عبارت دیگر داریم:

Randomly initialize K cluster centroid $\mu_1, ..., \mu_k \in R^n$
Repeate{

for $i = 1$ to m
$c^{(i)}$ = index (from i to k) of cluster centroid closest to $x^{(i)}$
$(c^{(i)} = argmin \left\|x^{(i)} - \mu_k\right\|^2)$
(minimize J() with respect to $c^{(1)}, ..., c^{(m)}$)

for $k = 1$ to K
μ_k = average (mean) of points assigned to cluster k
(minimize J() with respect to $\mu_1, ..., \mu_k$)

}

تابع هزینه ای که در الگوریتم بالا اشاره شده همان‌طور که گفته شد سعی در کوچک کردن فاصله هر نقطه centroid ای که آن داده به آن assign شده است دارد بنابر این داریم:

$$J(c^{(1)}, ..., (m), \mu_1, ..., \mu^k) = \sum_{i=1}^{m} \left\|x^{(i)} - \mu_c^{(i)}\right\|^2$$

$c^{(i)}$: index خوشه ای است که نمونه I ام در آن خوشه است.

cluster centroid k: μ_k $\mu^k \in R^n$

$\mu_c^{(i)}$: خوشه مربوط به داده iام است.

این الگوریتم در تعداد محدودی iteration حتماً همگرا می شود.

اثبات:

برای خوشه بندی n داده در k خوشه، حداکثر k^n حالت برای انجام آن وجود دارد. یعنی برای هر داده k تا انتخاب داریم. بعد از هر iteration اگر خوشه نسبت داده شده به هر کلاس تغییری نکرده بودند الگوریتم به پایان رسیده و اگر تغییر کرده‌اند به معنی این است که الگوریتم هنوزبه پایان نرسیده و هزینه این خوشه بندی جدید حتماً کمتر از حالت قبل است.

و از طرفی می‌دانیم که ممکن است الگوریتم global optimum نباشد و برای پیدا کردن بهترین local optimum تعداد خوشه ها را به ترتیب زیاد می‌کنیم و با روش‌هایی (مثل روش elbow method) بهترین تعداد خوشه را برای آن داده‌ها پیدا می کنیم. درواقع با افزایش تعداد خوشه ها حتماً cost کاهش میابد. بنابر این در بدترین حالت داده‌ها را به n خوشه تقسیم می‌کند.

۲) k_mean algorithm on Wine dataset

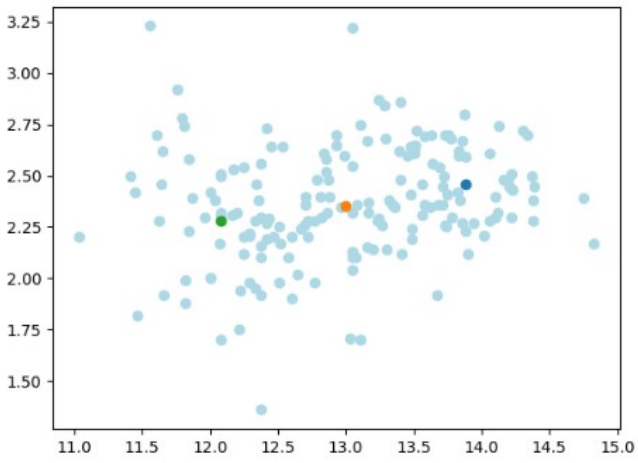
```
#%%
data = pd.read_csv('wine.data', header=None)
data = data.sample(frac=1).reset_index(drop=True)
y = data[0]
x1 = data.loc[:, 1].to_numpy().reshape(178, 1)
x2 = data.loc[:, 3].to_numpy().reshape(178, 1)
x = np.concatenate([x1, x2], axis=1)
m = x.shape[0]
```

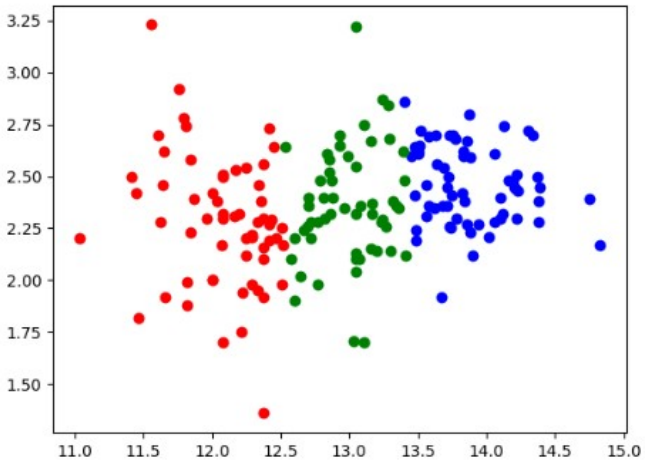
```
#%% initialize centroids
# data set has been shuffled
K = 5
centroids = np.zeros(shape=(K, x.shape[1]))
for k in range(K):
    centroids[k] = x[k]
```

```
#%%
c = np.zeros(x.shape[0])
for repeat in range(100):
    for i in range(m):
        c[i] = compute_min_distance(x[i], centroids)
    for k in range(K):
        centroids[k] = np.mean(x[c == k, :], axis=0)
```

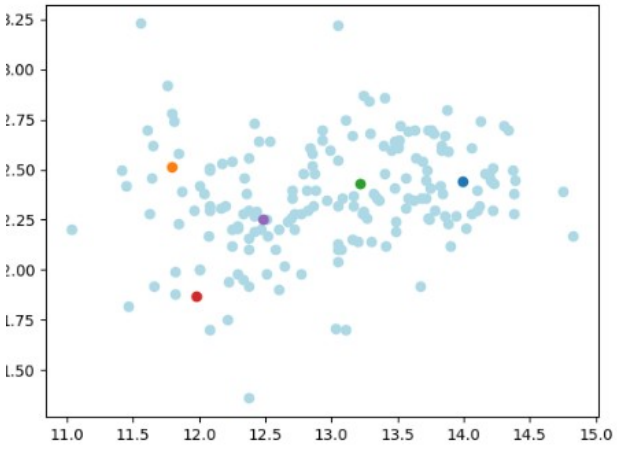
```
#%% compute J
distance = np.zeros(x.shape[0])
for k in range(K):
    distance[k] = np.square(np.linalg.norm(x[c == k] - centroids[k]))
J = np.sum(distance)
```

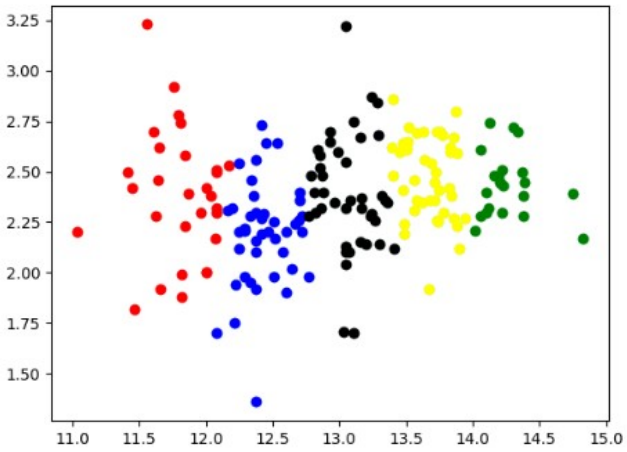
k=3


3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0

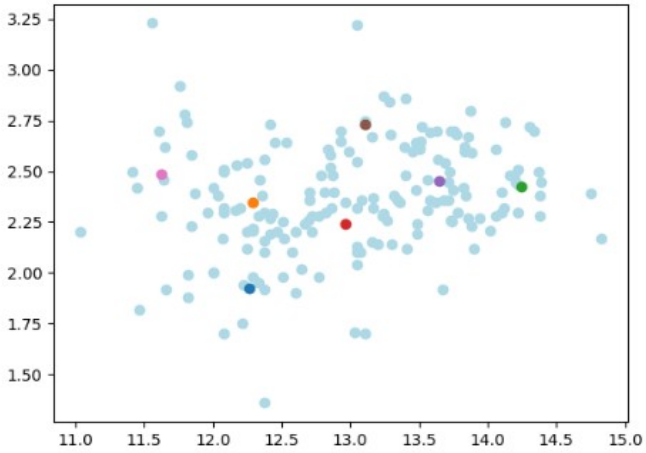

3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0

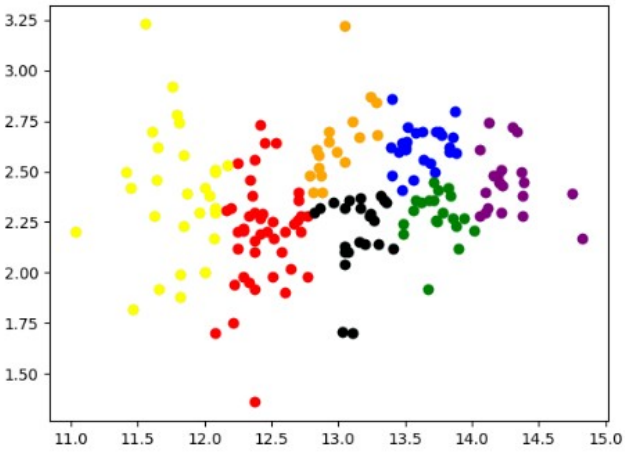
k=5


11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0


3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0

k=7


3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0


3.25
3.00
2.75
2.50
2.25
2.00
1.75
1.50
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0

۳) معیار های شباهت درونی و بیرونی

پس از خوشه بندی داده‌ها معیار هایی برای بررسی صحت آن وجود دارد.

معیار شباهت درونی: شباهت را برای داده‌های درون یک خوشه می سنجد. مثلاً تعداد یا چگالی و یا نزدیکی داده‌ها در هر خوشه را مورد ارزیابی قرار می دهد. چگالی داده‌ها یعنی داده‌ها در هر خوشه چقدر به یکدیگر نزدیک هستند و برای محاسبه آن معمولاً از ماتریس کوواریانس استفاده می‌کنند که هر چه کوچک‌تر باشد به این معنی است که داده‌ها به هم نزدیک‌تر هستند.

در ادامه دو معیار شباهت درونی را بررسی می کنیم:

۱) *Root-mean-square standard deviation (RMSSTD)*

این معیار مقدار ریشه میانگین مربعات درون هر خوشه محاسبه می کند.

$$SS = \sum_{i=1}^{n}(x_i - \overline{x})^2$$

$$V_{RMSSTD} = (\frac{SS_w}{p(n-k)})^{\frac{1}{2}}$$

و مقادیر بزرگ RMSSTD به معنی این است که نمونه‌های درون یک خوشه همگن(homogeneous) نیستند.

بنابر هر چه این شاخص کوچک‌تر باشد به معنی این است که الگوریتم بهتر عمل کرده و داده‌های هر خوشه به یکدیگر نزدیک می باشند.

۲) Dunn index

یکی دیگر از معیار های شباهت درونی می‌باشد به این صورت عمل می‌کند که:

برای هر خوشه فاصله بین نقاط درون آن خوشه و نقاط خوشه های دیگر را محاسبه می‌کند و کمترین این مقدار را به عنوان $min.separation$ در نظر می گیرد.

$$min.seperation = \min_{x \in C_i, y \in C_j} d(x,y)$$

و همچنین برای هر خوشه فاصله بین نقطه های درون هر خوشه را محاسبه می‌کند . بیشترین مقدار آن را به عنوان $max.diameter$ درنظر می گیرد.

$$max.diameter = \max_{x,y \in C_l} d(x,y)$$

و سپس معیار Dزیر را محاسبه می کند:

$$D = \frac{min.separation}{max.diameter}$$

هر چه مقدار این شاخص بزرگتر باشد، بیانگر تفکیک‌پذیری بهتر و در نتیجه خوشه‌بندی موثرتر است.

البته این معیار شباهت درونی را در نظر می‌گیرد ولی به صورت کلی ترکیبی از معیار درونی و بیرونی می باشد.

معیار شباهت بیرونی: شباهت بین داده‌های دو خوشه را می سنجد.

۱) Purity Index (شاخص خلوص)

یک معیار شبات بیرونی است که درصد مطابقت بین برچسب‌های خوشه‌بندی و برچسب‌های واقعی را می‌سنجد. به این صورت که نسبت تعداد نقاطی را که خوشه آن‌ها درست تشخیص داده شده را به نسبت کل نقاط آن خوشه محاسبه می کند.

حداکثر مقدار این شاخص یک می‌باشد و هنگامی اتفاق می‌افتد که همه نقاط یک خوشه واقعاً متعل به همان خوشه باشند.

$$Purity(S,C) = \frac{\sum_m \max_n |S_m \cap C_n|}{N}$$

۲) Rand index

برای نشان دادن میزان شباهت بین دو شیوه برچسب‌گذاری از این روش استفاده می شود.

اگر یک ست از نقاط $S = o_1, ..., o_n$ داشته باشیم و آن‌ها را به دو دسته تقسیم کنیم.

A: تعداد زوج‌هایی که هم در خوشه‌ها دارای برچسب یکسانی هستند و هم برچسب دسته‌ها برای آن‌ها یکسان است.

B: تعداد زوج‌هایی که برچسب خوشه‌هایشان متفاوت است و البته برچسب دسته‌های متفاوتی نیز دارند.

$$\text{Rand(S,C)} = \frac{A+B}{N(N-1)/2}$$

این روش را به صورت زیر هم می‌توان بازنویسی کرد:

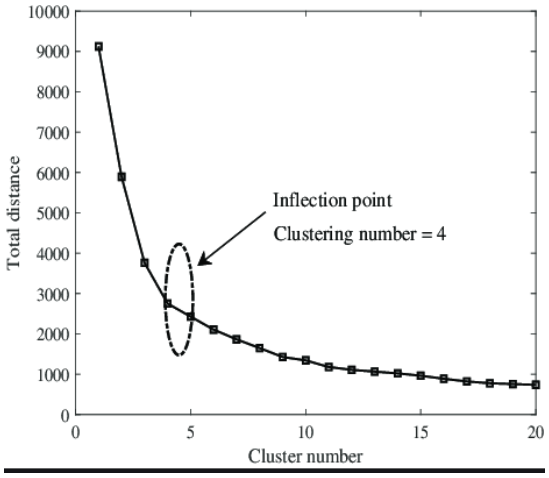
$$RI = \frac{TP+TN}{TP+FP+FN+TN}$$

۵) روش بهینه پیدا کردن تعداد خوشه بندی:

انتخاب k مناسب می‌تواند بر اساس نیاز و یا با مشاهده داده‌ها و به صورت چشمی صورت بگیرد ولی روش مطمین تر آن استفاده از elbow method است.

elbow method:

در این روش k را یک قرار می‌دهیم و با افزایش آن به تدریج در هر مرحله تابع خطا را محاسبه می کنیم. با افزایش تعداد خوشه ها خطا همواره کاهش می‌یابد ولی بهینه ترین مقدار برای k آن مقداری است که خطا از آن مقدار به بعد به کندی کاهش می‌یابد و یا به عبارت دیگر در نمودار آن به یک نقطه بازویی بر می خوریم.



سؤال دوم) Hierarchical clustering

در این روش نقاط با کمترین فاصله (بیشترین شباهت) با هم ادغام می‌شوند و در هر مرحله یکی از تعداد خوشه ها کم می شود.

اگر ماتریس D که میزان شباهت را نشان می‌دهد به صورت زیر تعریف کنیم، الگوریتم آن به این صورت می شود:

$$D_{(m*m)=Dist(X^{(i)},X(j))}$$

درواقع ماتریس D فاصله بین هر دو داده i و j را نشان می دهد.

به عنوان مثال x^i , x^j کمترین فاصله را دارند:

۱) سطر و ستون I و j از ماتریس D حذف می‌شود.

۲) به جای آن یک سطر و ستون جدید اضافه می‌شود . که ادغامی از آن دو داده حذف شده می باشد.

۳) معیار توقف :

- max treshold روی فاصله بین دو خوشه ادغام شده
- min treshold تعداد خوشه ها

در این سؤال به دلیل اینکه می‌خواهیم دیندوگرام رسم کنیم تا زمانی که به یک خوشه برسیم الگوریتم را ادامه می هیم.

ابتدا معیار شباهت SMC , JC را برای تک تک داده‌ها محاسبه می کنیم:

$$SMC(x_i,x_j)=\frac{n_{11}+n_{00}}{n_{00}+n_{11}+n_{01}+n_{10}}$$

$$JC(x_i,x_j)=\frac{n_{11}}{n_{11}+n_{01}+n_{10}}$$

SMC

	0	1	2	3	4	5
0	1.00000	0.60000	0.80000	0.40000	0.60000	0.40000
1	0.60000	1.00000	0.40000	0.80000	0.20000	0.40000
2	0.80000	0.40000	1.00000	0.60000	0.40000	0.60000
3	0.40000	0.80000	0.60000	1.00000	0.00000	0.60000
4	0.60000	0.20000	0.40000	0.00000	1.00000	0.40000
5	0.40000	0.40000	0.60000	0.60000	0.40000	1.00000

JC

	0	1	2	3	4	5
0	1.00000	0.50000	0.66667	0.25000	0.50000	0.25000
1	0.50000	1.00000	0.25000	0.66667	0.20000	0.25000
2	0.66667	0.25000	1.00000	0.33333	0.25000	0.33333
3	0.25000	0.66667	0.33333	1.00000	0.00000	0.33333
4	0.50000	0.20000	0.25000	0.00000	1.00000	0.25000
5	0.25000	0.25000	0.33333	0.33333	0.25000	1.00000

در single linkage در هر مرحله کمترین میزان شباهت بین هر دو داده را برای داده ادغام شده در نظر می گیریم.

ابتدا برای SMC داریم:

۱) در مرحله اول همان‌طور که مشخص است کمترین شباهت را داده ۳ و ۴ دارند که معیار شباهت برای آن‌ها صفر است بنابر این این دو سطر و ستون را حذف می‌کنیم و به جای آن سطر و ستون ادغام شده که کمترین عدد برای هر کدام از آن‌ها است را قرار می دهیم. مثلاً برای پیدا کردن خانه ۰ و (۳و۴) در جدول جدید. بین دو مقدار عدد D(0,3)=0.4 و D(0,4)=0.6 کمترین مقدار که 0.4 است را انتخاب می کنیم.

	0	1	2	5	3,4
0	1				
1	0.6	1			
2	0.8	0.4	1		
5	0.4	0.4	0.6	1	
3,4	0.4	0.2	0.4	0.4	1

در این مرحله کم‌ترین شباهت بین داده ۱ و (۳و۴) می باشد.

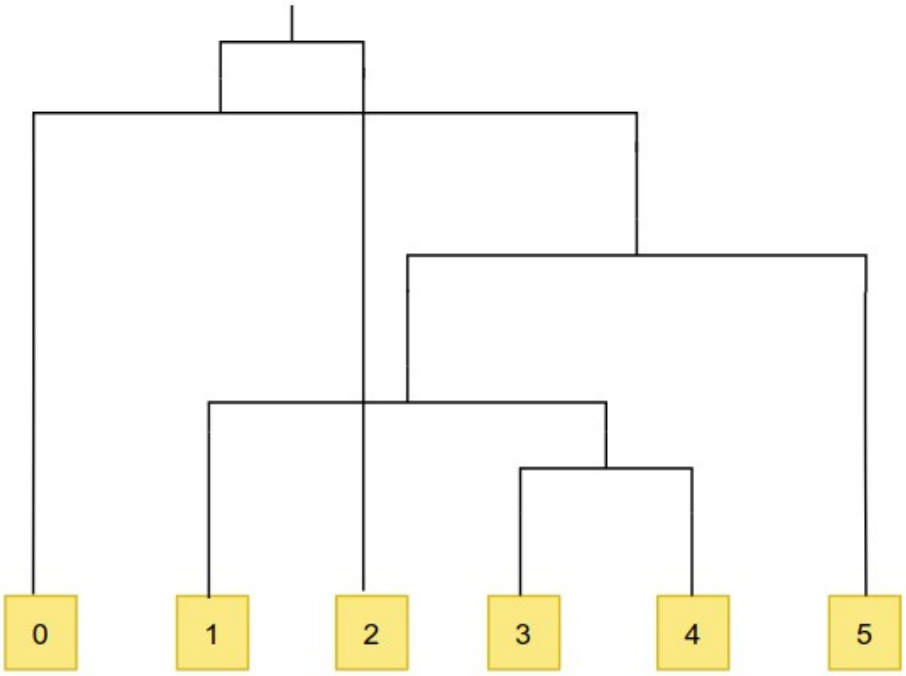
	0	2	5	1,3,4
0	1			
2	0.8	1		
5	0.4	0.6	1	
1,3,4	0.4	0.4	0.4	1

در این مرحله کمترین مقدار ۰.۴ است یکی از آن‌ها را انتخاب می کنیم.

	0	2	1,3,4,5
0	1		
2	0.8	1	
1,3,4,5	0.4	0.4	1

	0,2,3,4,5	2
0,2,3,4,5	1	0.4
2	0.4	1

برای نمودار دندوگرام به ترتیب نشان می‌دهیم که در هر مرحله کدام دو داده ادغام شده اند.

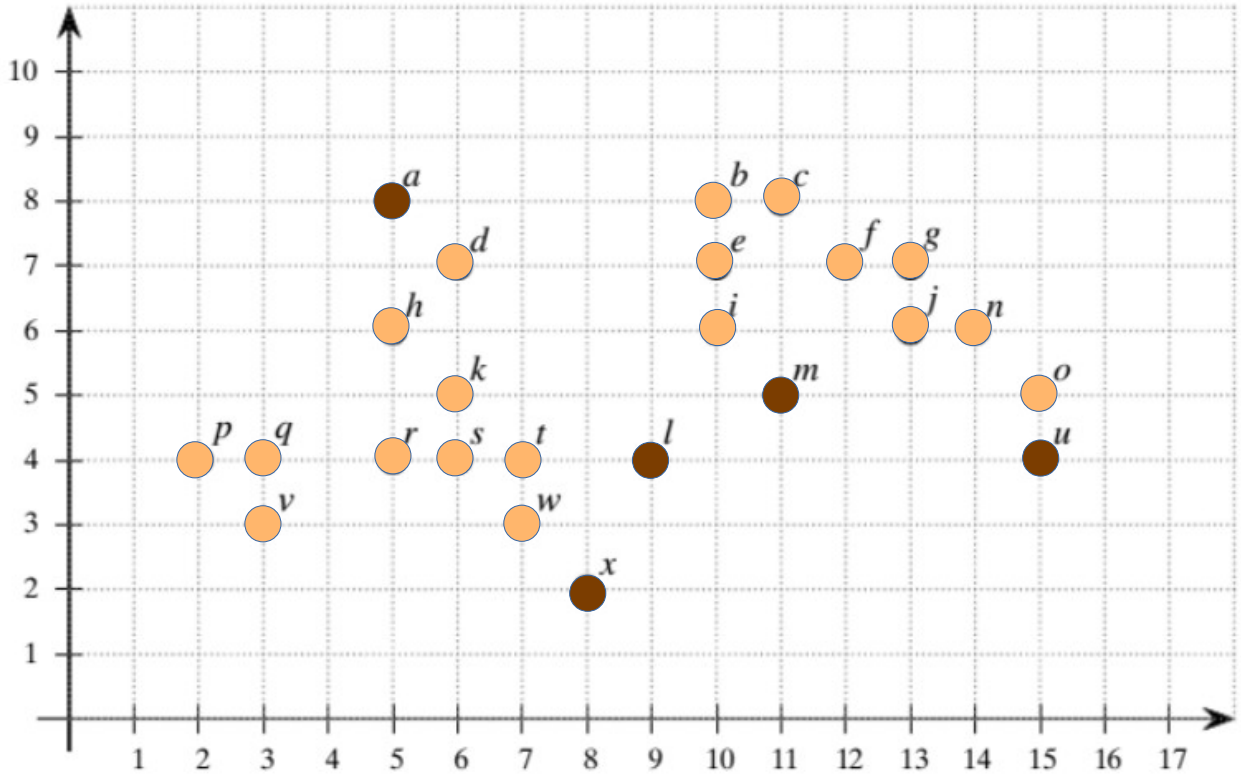


سوال۳) DBSCAN

نقاط مرکزی (core points) : نقاطی که به تعداد حداقل minPts که در این سؤال ۳ می‌باشد، نقطه در همسایگی ϵ =2 از خود داشته باشد. (در این شمارش خود نقطه هم حساب است و نقاط روی مرز را هم در نظر می گیریم.)

نقاط مرزی: نقاطی که به تعداد کمتر از minPts در شعاع ϵ از خود نقطه وجود دارد ولی حداقل یک نقطه مرکزی در فاصله ϵ از خود دارد.

نقاط مرکزی با رنگ نارنجی و نقاط مرزی با رنگ قهوه ای مشخص شده اند.



در این مثال o در همسایگی خود u , n را دارد که مجموعاً ۳ نقطه می‌شود. بنابر این نقطه مرکزی حساب می‌شود.

u در فاصله ۲ از خودش فقط نقطه o را دارد بنابر این از ۳ کمتر نقطه در همسایگی اش دارد ولی به دلیل اینکه o نقطه مرکزی است، u نقطه مرزی به حساب می اید.

الگوریتم DBSCAN:

ابتدا نقطه o را در نظر می گیریم، فاصله همه نقاط تا o را محاسبه می کنیم.

همان‌طور که مشخص است نقاط n, u , o فاصله کمتر از ۲ با نقطه o دارند.

بنابر این o نقطه مرکزی است.

	x	y	dis(o, X_i)
a	5.00000	8.00000	10.44031
b	10.00000	8.00000	5.83095
c	11.00000	8.00000	5.00000
d	6.00000	7.00000	9.21954
e	10.00000	6.00000	5.09902
f	12.00000	7.00000	3.60555
g	13.00000	7.00000	2.82843
h	5.00000	6.00000	10.04988
i	10.00000	6.00000	5.09902
j	13.00000	6.00000	2.23607
k	6.00000	5.00000	9.00000
l	9.00000	4.00000	6.08276
m	11.00000	5.00000	4.00000
n	14.00000	6.00000	1.41421
o	15.00000	5.00000	0.00000
p	2.00000	4.00000	13.03840
q	3.00000	4.00000	12.04159
r	5.00000	4.00000	10.04988
s	6.00000	4.00000	9.05539
t	7.00000	4.00000	8.06226
u	15.00000	4.00000	1.00000
v	3.00000	3.00000	12.16553
w	7.00000	3.00000	8.24621
x	8.00000	2.00000	7.61577

سپس u را بررسی می کنیم:

برای u فقط نقطه o , u فاصله کمتر از ۲ دارند که o هم نقطه مرکزی است بنابر این u به عنوان نقطه مرزی تشخیص داده می شود.

	x	y	dis(u, X_i)
a	5.00000	8.00000	10.77033
b	10.00000	8.00000	6.40312
c	11.00000	8.00000	5.65685
d	6.00000	7.00000	9.48683
e	10.00000	6.00000	5.38516
f	12.00000	7.00000	4.24264
g	13.00000	7.00000	3.60555
h	5.00000	6.00000	10.19804
i	10.00000	6.00000	5.38516
j	13.00000	6.00000	2.82843
k	6.00000	5.00000	9.05539
l	9.00000	4.00000	6.00000
m	11.00000	5.00000	4.12311
n	14.00000	6.00000	2.23607
o	15.00000	5.00000	1.00000
p	2.00000	4.00000	13.00000
q	3.00000	4.00000	12.00000
r	5.00000	4.00000	10.00000
s	6.00000	4.00000	9.00000
t	7.00000	4.00000	8.00000
u	15.00000	4.00000	0.00000
v	3.00000	3.00000	12.04159
w	7.00000	3.00000	8.06226
x	8.00000	2.00000	7.28011

	x	y	dis(n, X_i)
a	5.00000	8.00000	9.21954
b	10.00000	8.00000	4.47214
c	11.00000	8.00000	3.60555
d	6.00000	7.00000	8.06226
e	10.00000	6.00000	4.00000
f	12.00000	7.00000	2.23607
g	13.00000	7.00000	1.41421
h	5.00000	6.00000	9.00000
i	10.00000	6.00000	4.00000
j	13.00000	6.00000	1.00000
k	6.00000	5.00000	8.06226
l	9.00000	4.00000	5.38516
m	11.00000	5.00000	3.16228
n	14.00000	6.00000	0.00000
o	15.00000	5.00000	1.41421
p	2.00000	4.00000	12.16553
q	3.00000	4.00000	11.18034
r	5.00000	4.00000	9.21954
s	6.00000	4.00000	8.24621
t	7.00000	4.00000	7.28011
u	15.00000	4.00000	2.23607
v	3.00000	3.00000	11.40175
w	7.00000	3.00000	7.61577
x	8.00000	2.00000	7.21110

سپس n را بررسی می کنیم:

نقاط g, j , n, o در شعاع ۲ از nقرار دارند بنابر این n نقطه مرکزی است.

به این ترتیب الگوریتم را ادامه می دهیم.

نقاط نویزی همان نقاط outlier هستند. و نقاط outlier نقاطی هستند که نه نقطه مرکزی و نه مرکزی باشند.

در این مثال همه نقاط یا مرکزی و یا مرزی هستند بنابر این نقاط نویزی نداریم.

سؤال چهارم)

۱) کاهش بعد داده‌های Iris با استفاده از PCA:

ابتدا داده‌ها را لود می کنیم.

```
#%% load data
data = pd.read_csv('iris.data', header=None)
data[4] = data[4].replace({"Iris-virginica": 0, "Iris-versicolor": 1, "Iris-setosa": 2})
data = data.sample(frac=1).reset_index(drop=True)
x = data.loc[:, 0:3].to_numpy()
y = data.loc[:, 4].to_numpy()
```

در PCA داده‌ها باید نرمال شوند که اگر scale هر feature متفاوت بود تأثیری در نتیجه نداشته باشد و همچنین میانگین هر feature را صفر می کند.

```
#%% pre processing
for col in range(x.shape[1]):
    x[:, col] = (x[:, col] - x[:, col].mean()) / x[:, col].std()
```

ماتریس کواریانس را به دست می آوریم.

$$cov = \frac{1}{m}\sum_{i=1}^{n}(X^{(i)})(X^{(i)})^{.T}$$

```
#%% compute sigma
cov = np.cov(x.T)
```

بردار ویژه و مقادیر ویژه را به دست می آوریم.

```
#%% compute eigen vector
u, s, vh = np.linalg.svd(cov, full_matrices=True)
```

K=2 تا از ستون‌های اول ماتریس بردار ویژه را انتخاب می کنیم.

سپس ویژگی‌های کاهش بعد یافته را با استفاده از ضرب داخلی x در $U_{reduced}$ را به دست می آوریم.

```
#%% compute Z
u_reduce = u[:, :2]
Z = np.zeros(shape=(x.shape[0],  2))
Z = np.dot(x, u_reduce)
```

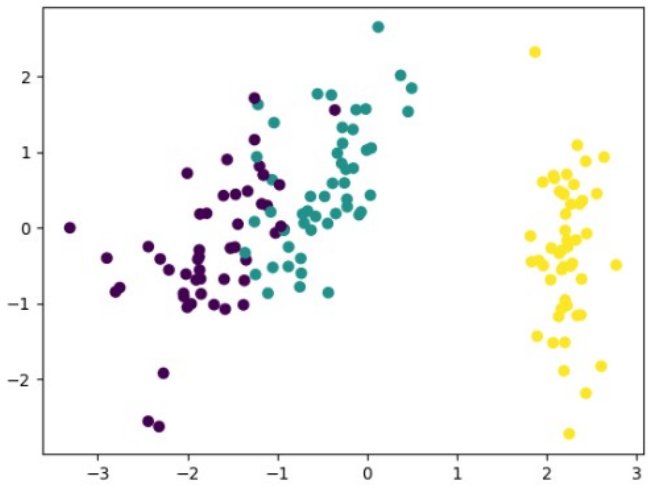
و داده‌ها را رسم می کنیم.

```
#%% plot data
plt.figure()
plt.scatter(Z[:, 0], Z[:, 1], c=y)
plt.show()
```

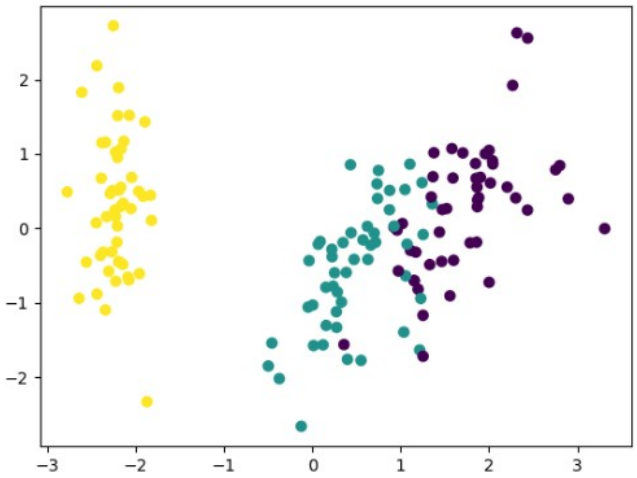
تقریبی از داده‌های اولیه را می‌توان به دست آورد.

```
#%% x approx
x_approx = np.zeros(shape = (x.shape[0], x.shape[1]))
for row in range(len(x)):
    x_approx[row] = np.dot(u_reduce, Z[row])
```

برای تصویر سازی بهتر، از داده‌های ستون ۴ برای مشاهده داده‌ها استفاده کرده ام. ولی در این سؤال این رنگ‌ها معنی نمی‌دهد و درست تر این بود که همه با یک رنگ رسم شوند به دلیل اینکه این لگوریتم صرفاً برای کاهش بعد و visualize کردن داده ها استفاده شده است.

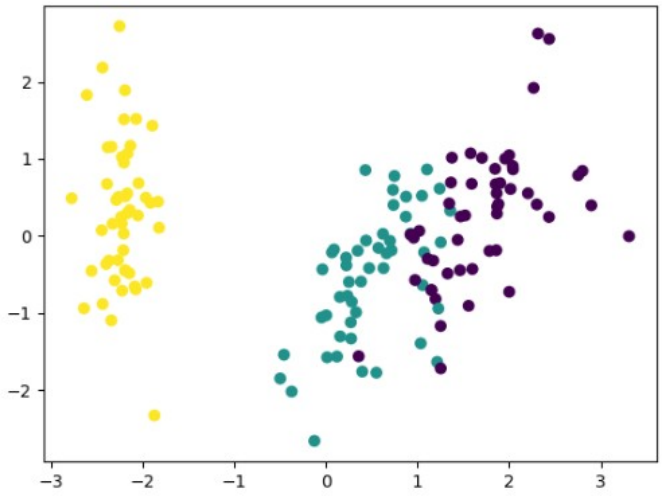


در فایل q4.1_sklearn برای مقایسه از کتابخانه‌های آماده استفاده کردم و خروجی آن به شکل زیر شد:



این دو تصویر قرینه یکدیگر هستند و این به این دلیل اتفاق افتاده است که در sklearn برای محاسبه ماتریس u کمی تفاوت دارد و اگر بردار های ویژه را در ۱- ضرب کنیم تصویر مشابه به دست خواهیم آورد:

تصویر با-u که مشابه همان حالتی است که از کتابخانه sklearn استفاده شده:



۲) کاهش بعد داده‌های Iris با استفاده از Power Method:

یک الگوریتم تکرار شونده (iterative) برای یافتن بزرگ‌ترین مقدار ویژه و بردار ویژه متناظر با آن است. مثلاً اگر ماتریس A_{n*n} را در نظر بگیریم که مقادیر ویژه $\lambda_1, \lambda_2, \ldots, \lambda_n$ دارد. و بردار های ویژه متناظر با آن $v_1, v_2, \ldots, v_n$ باشند.

و مقادیر ویژه آن به این صورت باشد: $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ در این روش هدف پیدا کردن λ_1 , λ_2 و بردار ویژه های متناظر با آن‌ها می باشد.

برای به دست آوردن بزرگترین مقدار ویژه با استفاده از الگوریتم زیر بزرگ‌ترین مقدار ویژه را به دست می آوریم:

for i=1:m

$$v^{(i)} = Av^{(i-1)}$$

$$v^{(i)} = \frac{v^{(i)}}{||v^{(i)}||}$$

و m تا جایی ادامه پیدا می‌کند که $||v^{(i)} - v^{(i-1)}|| < \epsilon$ باشد. درواقع تا جای یکه بردار ویژه دیگر تغییر نکند ادامه می دهیم.

```
def largest_eigenvector(Matrix):
    v = np.ones(shape=[4, 1])
    v = v / np.linalg.norm(v)
    largest_lambda = 0
    for i in range(10):
        v = np.dot(Matrix, v)
        largest_lambda = np.linalg.norm(v)
        v = v / largest_lambda
    return largest_lambda, v
```

بردار ویژه اول به صورت زیر در می آید:

```
#%%
A = np.cov(X.T)
largest_lambda, v1 = largest_eigenvector(A)
```

	0
0	0.36159
1	-0.08227
2	0.85657
3	0.35884

برای به دست آوردن دومین بزرگترین مقدار ویژه از چند روش می‌توان استفاده کرد. در روش shifted power method در ماتریس A که که دارای بزرگ‌ترین مقدار ویژه λ_1 است را در نظر می‌گیریم و اگر برای ماتریس $[A - \lambda_1 I]x$ به دست آوریم، داریم:

$$[A - \lambda_1 I]x = \alpha x$$

که به دلیل اینکه $0, \lambda_2 - \lambda_1, \lambda_3 - \lambda_1, \ldots, \lambda_n - \lambda_1$ مقادیر ویژه ماتریس دوم بر حسب ماتریس اولیه است، $\alpha = \lambda_2 - \lambda_1$ می باشد.

اما روش تقلیل توانی روش بهینه تری استرت که:

باید λ_1 را از ماتریس قبلی حذف کنیم و بزرگ‌ترین مقدار ویژه را برای ماتریس جدید به دست آوریم.

می‌دانیم که اگر $B = P^{-1}AP$ باشد در این صورت: $\lambda(A) = \lambda(B)$

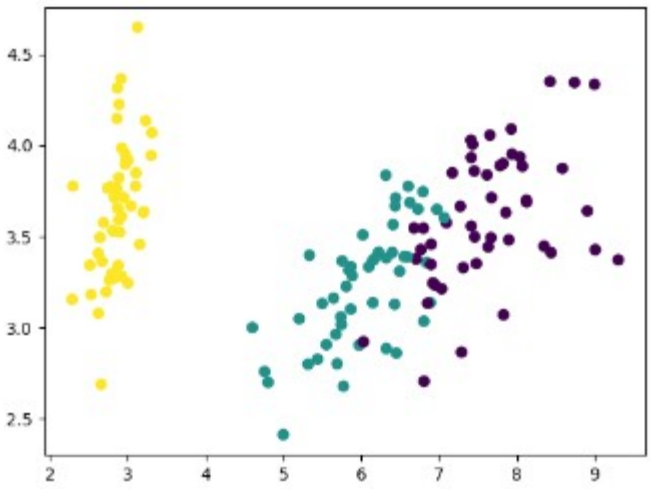
در این سؤال از روش اول استفاده می کنیم.

```
#%%
B = A - np.eye(4)*largest_lambda
second_largest_lambda, v2 = largest_eigenvector(B)
```

	0
0	0.21627
1	0.80712
2	-0.22393
3	0.50164

و مقادیر کاهش بعد یافته را به دست می آوریم.

```
#%%
v = np.concatenate((v1, v2), axis=1)
Z = np.zeros(shape=(X.shape[0],  2))
Z = np.dot(X, v)
```



۳) کاهش بعد با استفاده از کرنل غیر خطی:

پس از لود کردن داده‌ها و نرمال کردن آن‌ها، مدل را با استفاده از کرنل غیر خطی آموزش می‌دهیم و داده‌های کاهش بعد یافته را به دست آوریده و رسم می کنیم.

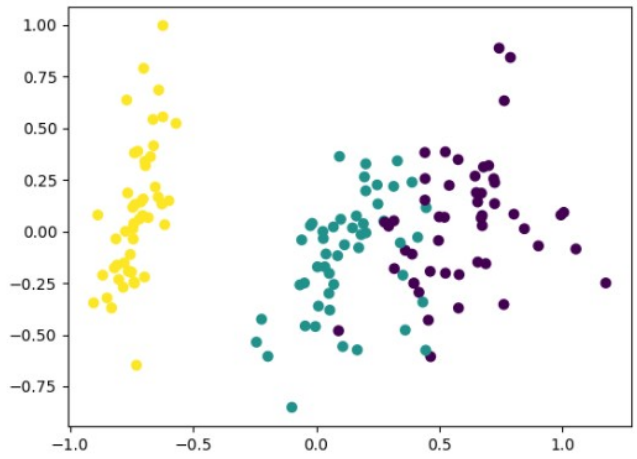
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import KernelPCA

data = pd.read_csv('iris.data', header=None)
data = data.sample(frac=1).reset_index(drop=True)
data[4] = data[4].replace({"Iris-virginica": 0, "Iris-versicolor": 1, "Iris-setosa": 2})
X = data.loc[:, 0:3].to_numpy()
y = data.loc[:, 4].to_numpy()

for col in range(X.shape[1]):
    X[:, col] = (X[:, col] - X[:, col].mean()) / X[:, col].std()

transformer = KernelPCA(n_components=2, kernel='sigmoid')
Z = transformer.fit_transform(X)

plt.figure()
plt.scatter(Z[:, 0], Z[:, 1], c=y)
plt.show()
```

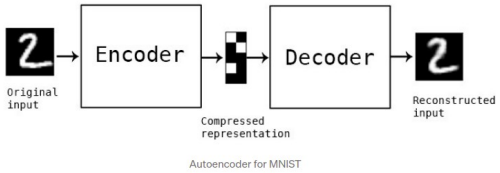


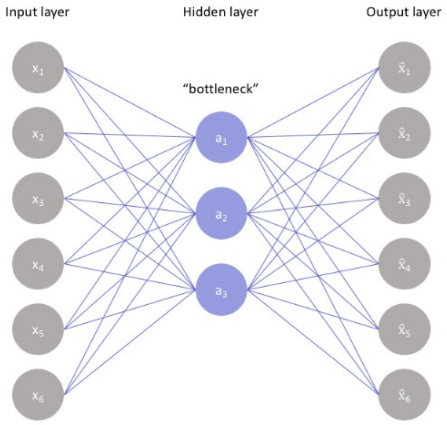
همان‌طور که مشاهده می‌شود با حالت اول که خطی بود تفاوت چندانی ندارد. که احتمالاً به دلیل این است که از کرنل سیگموید استفاده کرده ام.

۴) Auto Encoders:

autoencoder ها یک شبکه عصبی غیر نظارتی است که برای فشردسازی و encode کردن داده‌ها و سپس بازیابی و decode کردن از روی داده‌های encode شده می باشد. و سعی در این است که این داده‌های بازیابی شده بیشترین شباهت را به داده‌های اصلی داشته باشند.

Autoencoder ها با استفاده از در نظر نگرفتن نویز ها، بعد داده‌ها را کاهش می‌دهند.





Encoder: کاهش بعد داده‌ها و تبدیل آن‌ها به فرمت encode شده

Decoder: بازیابی داده‌ها به صورتی که بیشترین شباهت را به داده‌های اصلی داشته باشند.

Bottleneck: لایه‌ای که داده‌های فشرده شده را شامل می شود.

Reconstruction loss: معیاری برای مقایسه شباهت بین داده‌های بازیابی شده و داده‌های اصلی.

شبکه عصبی استفاده شده برای auto encoder ها می‌تواند انواع مختلفی داشته باشد از جمله: simple FeedForward , Convolutional Neural Network

برای فشردسازی تصویر روش‌های ساده مثل JPEG معمولاً عمل‌کرد بهتری دارند و از autoencoderها معمولاً برای داده‌های دیگر استفاده می‌شود ولی باید توجه داشت که autoencoder ها فقط هنگامی که داده جدید مشابه با داده‌ای که با آن مدل train شده مورد استفاده قرار می‌گیرد.

کاربرد Autoencoder ها : اگر چه برای فشردسازی تصویر هم می‌توان استفاده کرد ولی همان‌طور که گفته شد روش‌های بهتری هم برای این کار وجود دارد. امروزه دو کاربرد مهم autoencoder ها به شرح زیر است:

۱) data denoising: کاهش نویز در داده ها

۲) dimensionality reduction for data visualization : کاهش بعد داده‌ها به منظور مشاهده و تصویر سازی آن ها

این مدل با استفاده از کاهش خطای داده اصلی و داده بازیابی شده train می شود. به همین دلیل است که این مدل، مهم‌ترین ویژگی‌های ورودی را نگه می‌دارد و در کاهش بعد آن‌ها را از دست نمی دهد، چون در بازیابی داده‌ها به آن‌ها احتیاج دارد. و این کار در راستای کاهش خطا است.

نکته: اگر در این روش از تابع activation خطی استفاده کنیم جواب در نهایت مشابه روش کاهش بعدPCA می شود.

مساله مهم در این مدل(و اکثر مدل ها) این است که مدل به اندازه کافی خوب train شده باشد و خطا کم باشد و از طرفی overfitting هم اتفاق نیافتد. بنابر این علاوه بر تابع خطایی که تعریف می‌کنیم ترم پایدارسازی هم به آن اضافه می کنیم:

$$J(x, \hat{x}) + regularizer$$

که برای خطای MSE داریم:

$$J(x, \hat{x}) = \frac{1}{m} \sum (x - \hat{x})^2$$

و سعی در کاهش این خطا داریم.

۵)مقایسه الگوریتم LDA و PCA:

PCA الگوریتم غیر نظارتی یا unsupervised است و در این مسائل مقدار y را نداریم ولی الگوریتم LDA برای مسائل نظارتی یا supervised استفاده می‌شود در این مسائل مقدار y را داریم. در هر دو الگوریتم سعی در کاهش بعد داریم ولی در LDA تلاش می‌کند که تمایز بین کلاس‌های مختلف را بعد از کاهش بعد، حفظ کند. اما PCA صرفا خطای تصویر سازی را حداقل می‌کند.

اگر تعداد کلاس‌ها را c در نظر بگیریم:

$$y^{(i)} \in 1, 2, ..., c$$

در LDA تعداد کلاس‌ها را حداکثر به c-1 می‌توان کاهش داد در صورتی که در PCA این محدودیت را نداریم. چون اصلاً کلاس‌ها را نداریم و یادگیری غیر نظارتی است.

بنابر این در PCA کاهش بعد از n به k است به صورتی که k<n باشد. و در LDA از n به حداکثر c-1 کاهش بعد داریم.

در LDA تصویر سازی داده‌ها به صورتی است که فاصله کلاس‌ها از هم زیاد تر باشد در حال که پراکندگی یا واریانس داده‌ها در درون هر کلاس کم باشد و داده‌های درون هر کلاس به هم شبیه تر باشند.

البته می‌توان از PCA برای سرعت بخشیدن به الگوریتم های supervisedهم استفاده کرد.