

به نام خدا

دانشگاه صنعتی شریف - دانشکده مهندسی کامپیوتر

مقدمه‌ای بر یادگیری ماشین

فصل دوم: مرور روش‌های کلاسیک یادگیری ماشین

مبحث: خوشه‌بندی

نویسنده: حدیث احمدیان

فهرست مطالب

• 1- مقدمه

◦ 1-1. یادگیری بدون ناظر

◦ 1-2. خوشه‌بندی

◦ 1-2-1. انواع خوشه‌بندی

◦ 1-2-2. تعریف خوشه

◦ k-means - 2

◦ 2-1. معرفی الگوریتم

◦ 2-2. پیاده‌سازی k-means

◦ 2-3. مقداردهی اولیه مرکز خوشه‌ها

◦ 2-4. بهبودهای k-means

◦ ++K-means.2-4-1

◦ K-means.2-4-2 تسریع شده

◦ mini-batch K-means.2-4-3

◦ 2-5. انتخاب تعداد خوشه‌ها

◦ 2-5-1. استفاده از inertia

◦ 2-5-2. silhouette score

2-6. محدودیت‌های k-means

- کاربردهای خوش‌بندی

Image Segmentation .3-1

- 3-2. پیش‌پردازی داده‌ها

Semi-Supervised learning .3-3

- 3-4. یادگیری فعال (active learning)

1. مقدمه

1-1. یادگیری بدون ناظر (Unsupervised learning)

در دنیای امروزی اکثر کاربردهای یادگیری ماشین مبتنی بر یاگیری با ناظر (supervised learning) هستند. این بدين معنی است که برای اين کاربردها نیاز به داده‌های برچسب‌دار داریم یعنی برای آموزش مدل خود علاوه بر ویژگی‌ها باید برچسب داده‌های مورد نظر را نیز داشته باشیم حال آنکه داده‌های برچسب دار تنها بخش بسیار کوچکی از تمام داده‌های موجود هستند.

اجازه دهید با یک مثال بیشتر به این موضوع بپردازیم. فرض کنید در یک کارخانه که کالایی را تولید می‌کند وظیفه داریم کالای سالم از کالای خراب را تشخیص دهیم و برای انجام این کار می‌خواهیم یک مدل آموزش دهیم. ساخت داده بدون برچسب برای این کار بسیار راحت است. کافیست یک دوربین در اختیار داشته باشیم که از تمام کالاهای تولید شده در خط تولید عکسبرداری کند اما تنها با داشتن این عکس‌ها نمیتوانیم به صورت supervised عمل کنیم، زیرا نمی‌دانیم که هر عکس متعلق به کالای سالم است یا یک کالای خراب. به بیان دیگر برچسب متعلق به هر داده را در اختیار نداریم. برای برچسب گذاری داده‌ها نیاز به یک اپراتور انسانی داریم که تک تک عکس‌ها را بررسی کند و مشخص کند که آن کالا سالم بوده است یا خیر. می‌دانیم که این کار بسیار پرهزینه خواهد بود، به همین دلیل اغلب مجبور خواهیم بود تنها بخش کوچکی از داده‌های موجود را به طور تصادفی انتخاب کنیم و تنها آن بخش کوچک را برچسب گذاری کنیم.

اولین مشکل این است که به دلیل کوچک بودن مجموعه داده‌های برچسب دار، نمی‌توان مدل توانمندی را آموزش داد. علاوه بر این اگر بخش‌هایی از کالاهای تولیدی توسط کارخانه تغییر کند، دیگر مدل ما قابل استفاده نخواهد بود و باید از اول عکس‌های کالاهای جدید برچسب گذاری شود و مدل با داده‌های جدید آموزش ببیند.

یک مثال جالب برای مقایسه‌ی میزان داده‌های برچسب دار و بدون برچسب موجود، مثال کیک و گیلاس است. اگر داده‌های بدون برچسب موجود را به اندازه یک کیک در نظر بگیریم، داده‌های برچسب دار تنها به اندازه گیلاس کوچکی روی کیک است! پس مشخص است که اگر ما رویکردی در اختیار داشتیم که بدون نیاز به برچسب داده‌ها بتواند یک مدل را آموزش دهد مزیت بزرگی برای ما خواهد بود به این دلیل که تعداد داده‌های بدون برچسب در دسترس در حال حاضر بسیار بیشتر از داده‌های برچسب دار است و برچسب گذاری یک پروسه‌ی پرهزینه است.

به رویکردهایی که بدون نیاز به برچسب میتوانند از داده‌ها استفاده کنند روش‌های بدون ناظر یا unsupervised گفته می‌شود.

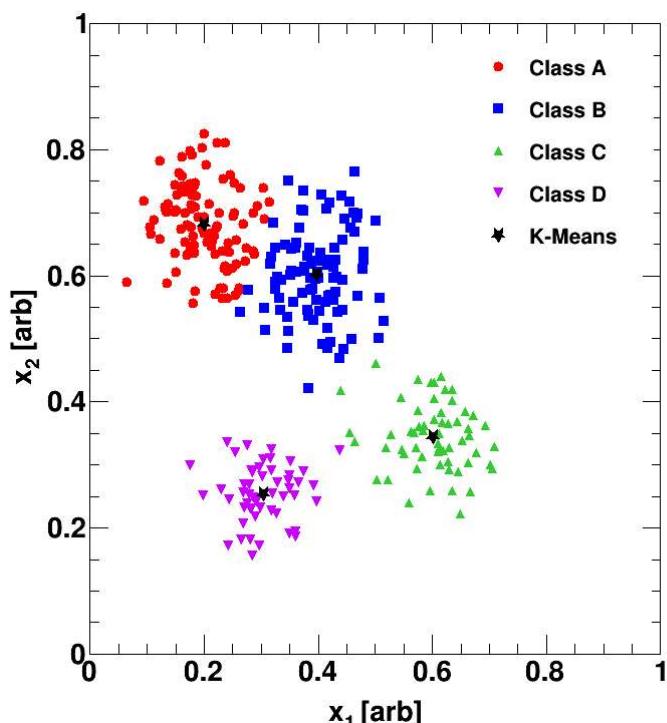
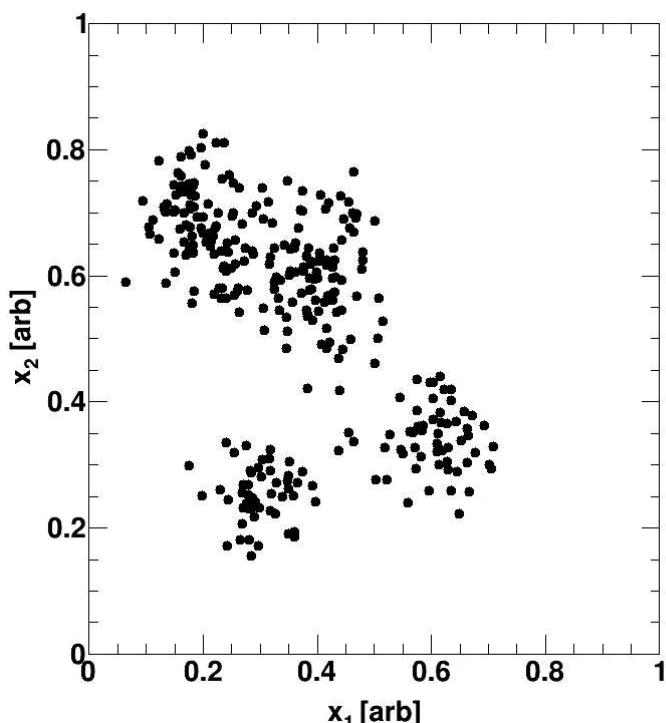


[pic source](#)

(Clustering) 1-2. خوشبندی

خوشبندی از رویکردهای Unsupervised است که هدف اصلی در آن، گروه‌بندی نمونه‌های است به صورتی که نمونه‌های مشابه در یک گروه قرار بگیرد. خوشبندی در کاربردهای زیادی استفاده می‌شود از این کاربردها می‌توان به سیستم‌های recommender، موتورهای جستجو، سگمنتیشن تصاویر، یادگیری semi-supervised، کاهش بعد و... اشاره کرد.

مثال زیر را در نظر بگیرید، در نمودار سمت راست داده‌ها برچسب دارند و یک رویکرد با ناظر می‌توانند از این داده‌ها استفاده کنند (مانند classification) اما همان نمونه‌ها در سمت چپ را بدون برچسب‌های ایشان مشاهده می‌کنید و هرچند این داده‌ها برچسب ندارند، اما باز هم جدا بودن بخشی از داده‌ها نسبت به سایر داده‌ها واضح است. این نشان می‌دهد بدون داشتن برچسب داده‌ها، ویژگی‌ها بدون داشتن برچسب‌ها می‌توانند اطلاعات کافی برای نسبت دادن هر نمونه به یک گروه را در اختیار ما قرار دهند که این همان رویکرد بدون ناظر است (مانند خوشبندی).



[pic source](#)

1-2-1. انواع خوشبندی :

الف) دسته بندی بر اساس نرم یا سخت بودن سخت (hard) : هر نمونه دقیقاً به یک خوش نسبت داده میشود. در واقع خروجی این است که هر نمونه دقیقاً متعلق به کدام خوش است.

نرم (soft) : هر نمونه با احتمالی بین ۰ و ۱ به چندین خوش نسبت داده میشود. در واقع خروجی این است که احتمال تعلق هر نمونه به هر کدام از خوشها چقدر است.

الف) دسته بندی بر اساس hierachial یا patitional بودن patitional : تمام خوشها در سطح یکسانی قرار دارند.

سلسه مراتبی (hierachial) : وقتی به شکل سلسه مراتبی با ادغام خوشها کوچکتر به خوشها بزرگتر میرسیم. ظریف خوش خود میتواند زیر مجموعه‌ی یک خوشی سطح بالاتر باشد و چند خوشی کوچکتر را در برداشته باشد. این یعنی خوشها سطح دارد و همه در سطح یکسانی نیستند.

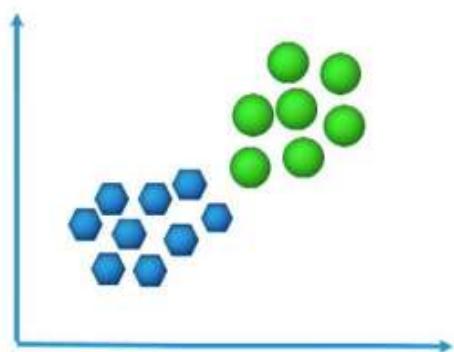
اگر در ابتدا هر نمونه را یک خوش در نظر بگیریم و سپس خوشها نزئیک را باهم ادغام کنیم تا نهایتاً به یک خوش برسیم این روش خوش بندی سلسه مراتبی به صورت تجمعی خواهد بود. رویکرد متفاوت میتواند این باشد که تمام داده‌ها یک خوش در نظر گرفته شود و سپس خوشها بزرگتر با تقسیم سلسه مراتبی به خوشها کوچکتر تبدیل شوند.

2-2-1. تعریف خوش:

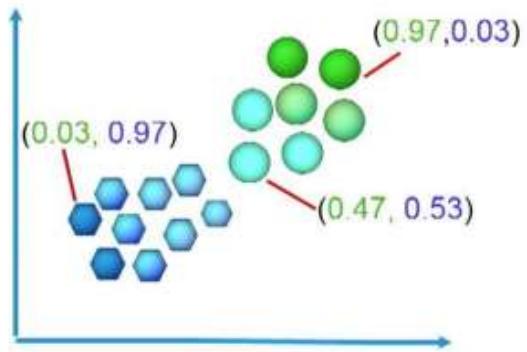
حال که با مفهوم و انواع خوشبندی آشنا شدیم شاید جالب باشد درمورد مفهوم خود خوش هم به طور دقیق‌تر صحبت کنیم. هیچ تعریف جهانی ای برای خوش وجود ندارد و تعریف خوش کاملاً به کاربرد ما بستگی دارد. بعضی از الگوریتم‌ها به دنبال داده‌هایی هستند که از یک نقطه‌ی مشخص به نام مرکز خوش، کمترین فاصله را داشته باشند. برخی به دنبال نواحی از نمونه‌ها هستند که چگالی بیشتری دارند و در آن‌ها خوشها هر شکلی می‌توانند داشته باشند. برخی دیگر از الگوریتم‌ها نیز به طور سلسه مراتبی خوشبندی می‌کنند و درواقع به دنبال خوش‌ای از خوشها هستند.

در ادامه یکی از مهم‌ترین الگوریتم‌های خوشبندی به نام k-means را مورد بررسی قرار می‌دهیم و سپس جزیات و کاربردهای بیشتر از خوشبندی را مرور می‌کنیم.

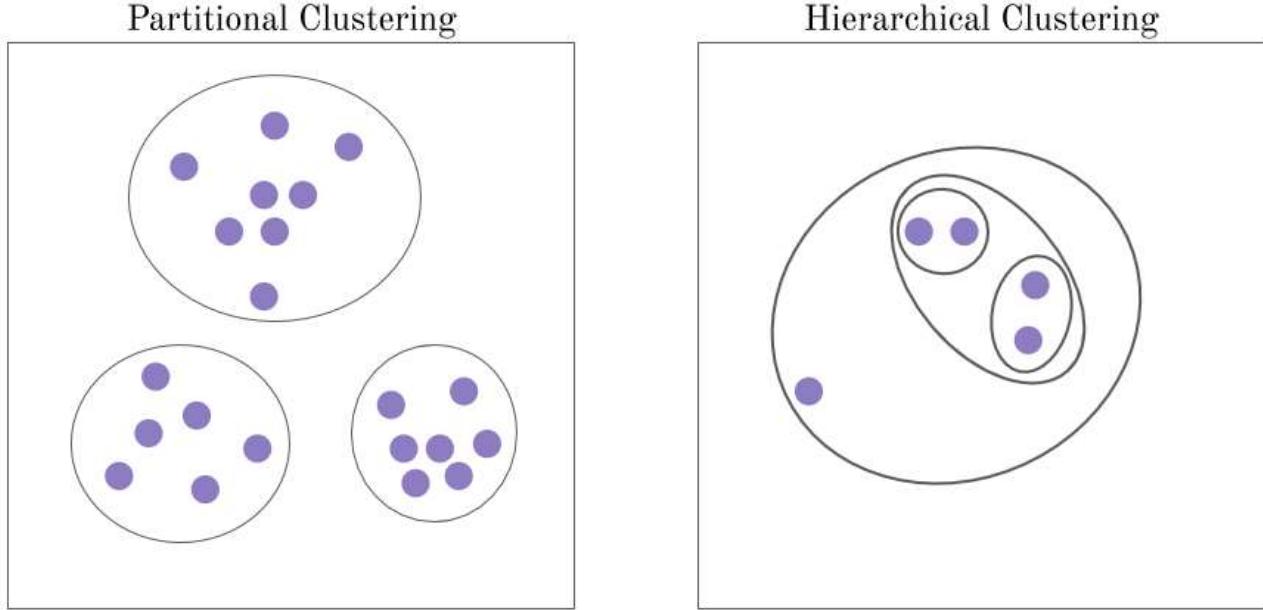
Hard Clustering



Soft Clustering



[pic source](#)



[pic source](#)

K-means.2

1-2. معرفی الگوریتم

این الگوریتم با فرض وجود k خوش، k مرکز خوش را مشخص می‌کند و هر داده را به خوش ای که داده به مرکز آن خوش نزدیک‌تر است نسبت می‌دهد.

به عبارت دیگر به ازای k مرکز خوشی c_k و x_n هایی که عضو هر خوش هستند هدف k-means کمینه کردن مقدار زیر است.

$$\sum_{n=1}^N \sum_{x_n \in c_k} \|x_n - c_k\|^2$$

فرض کنید مرکز خوشها را در اختیار داریم، بنابراین یافتن اینکه هر داده به کدام خوش متعلق است بسیار ساده است و کافی است ببینیم هر داده به کدام مرکز خوش نزدیک‌تر است.

حال فرض کنید مرکز خوشها را نداریم؛ ولی برچسب هر داده در اختیار ماست و میدانیم هر داده متعلق به چه خوش ای است، به این ترتیب می‌توانیم میانگین داده‌های هر خوش را به عنوان مرکز آن خوش معرفی کنیم.

اما در ابتدای امر ما نه مرکز خوشها را داریم و نه برچسب داده هارا پس رویکرد چه باید باشد؟ Kmeans به این گونه عمل می‌کند:

ابتدا k مرکز خوشی تصادفی انتخاب می‌کند (درواقع به طور تصادفی K تا از داده‌ها را به عنوان مراکز خوش انتخاب می‌کند)

سپس با استفاده از مرکزهای انتخاب شده، خوشی هر داده را مشخص می‌کنیم
سپس با استفاده از برچسب‌های مشخص شده مرکزهای جدید را محاسبه و به روز رسانی می‌کنیم
همین دو مرحله را تا زمانی ادامه می‌دهیم که مرکز خوشها دیگر جابجا نشوند.

نکته‌ی قابل این است که الگوریتم بعد از تعدادی مرحله (که اصولاً هم کم است) حتماً همگرا می‌شود و تا ابد نوسان نخواهد داشت. (این موضوع با این حقیقت قابل اثبات است که در هر مرحله mean squared distance

بین داده‌ها و نزدیک‌ترین مرکز خوش به آن‌ها تنها می‌تواند کاوش یابد)

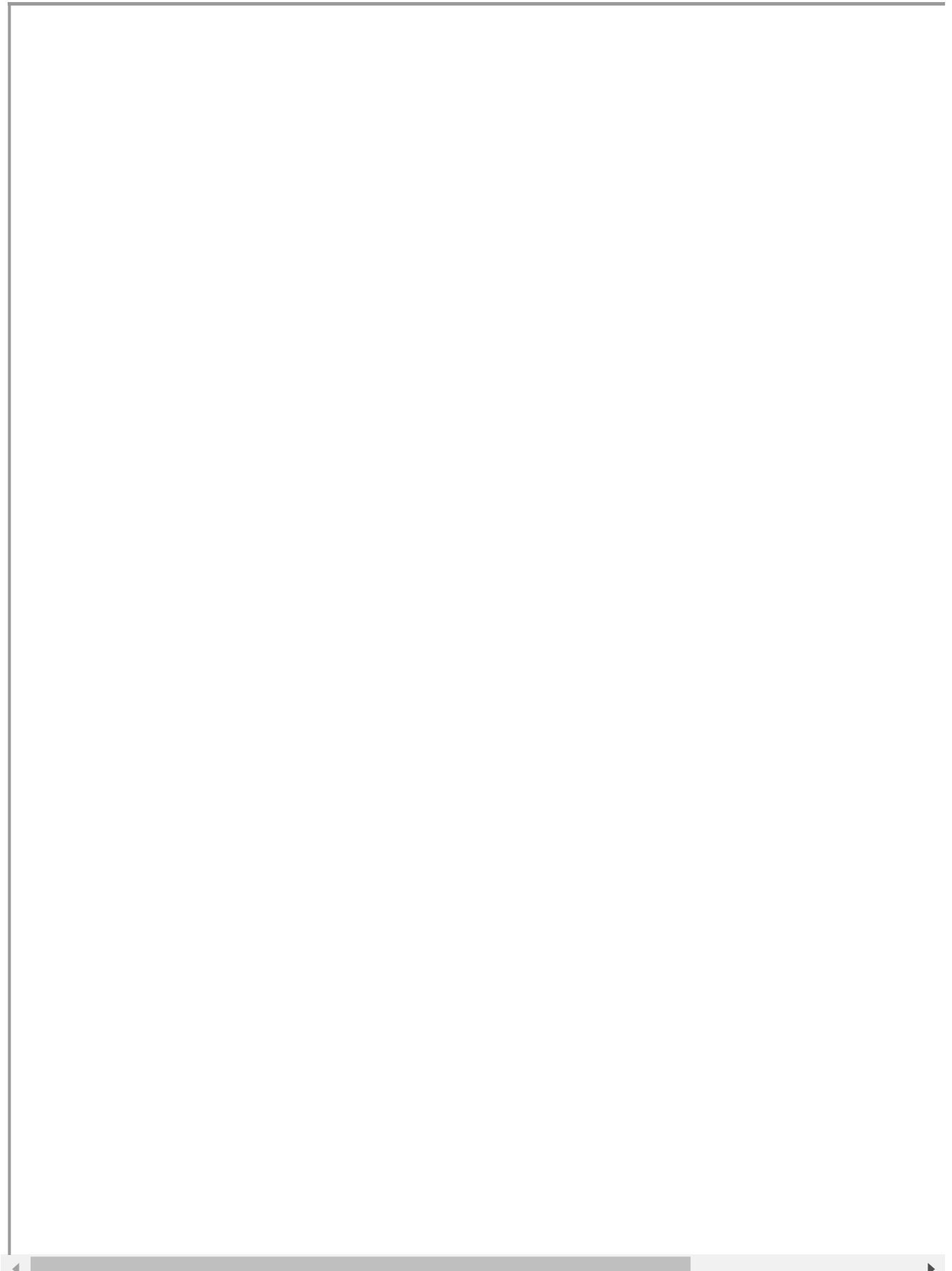
عملکرد الگوریتم k-means را میتوانید در مثال زیر بینید:

(اگر وب پیج در نوت بوک برای شما باز نمی‌شود، میتوانید مستقیماً از طریق لینک به وب پیج دسترسی پیدا کنید)

<http://shabal.in/visuals/kmeans/2.html>

%%html

```
<iframe src="http://shabal.in/visuals/kmeans/2.html" width="900" height="900"></iframe>
```



ابتدا اجازه دهید داره های مورد استفاده را معرفی کنیم. gene expression cancer RNA-Seq DataSet یک دیتاست است که ویژگی های آن میزان بیان ژن های مختلف در افراد مبتلا سرطان است که هر کدام نوع متفاوتی از تومور (BRCA, KIRC, COAD, LUAD and PRAD) را دارند.

لینک دیتابیس برای اطلاعات بیشتر

<https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq>:

نکته 1: دیتاست اصلی شامل تعداد 20531 ژن و میزان بیان هر کدام به ازای هر بیمار است (در واقع میزان بیان هر ژن یک ویژگی است پس دیتاست اصلی 20531 ویژگی به ازای هر نمونه دارد)، چون میخواهیم یک مثال ملموس داشته باشیم و داده ها قابل visulisation باشند، داده ها را با استفاده از TSNE کاهش بعد داده ایم اما این مراحل به دلیل اینکه مبحث مورد بحث ما نیست در این قسمت نیامده است. نتیجه ای نهایی پس از کاهش بعد یک دیتاست است که به ازای هر نمونه دو ویژگی دارد.

نکته 2: این دیتاست یک دیتاست با برچسب است اما ما قرار نیست از برچسب ها در طی خوش بندی استفاده کنیم چون خوش بندی یک رویکرد بدون ناظر است. بعد از خوش بندی بدون استفاده از برچسب ها، ما نتایج خوش بندی را با برچسب های واقعی مقایسه میکنیم تا این دید را منتقل کنیم که در کاربرد های واقعی نتیجه ای حاصل از خوش بندی میتواند یک نتیجه ای معنا دار باشد اما طبیعتاً در کاربرد های واقعی ما برچسب ها را نخواهیم داشت.

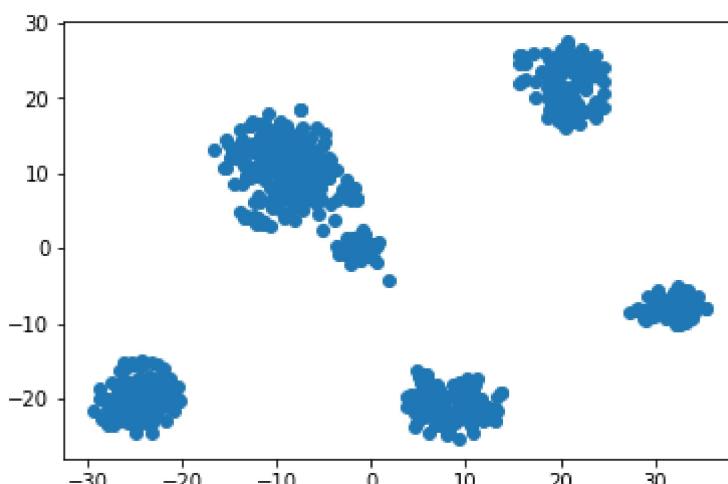
پس نهایتاً داده های مورد استفاده برای مثال خوش بندی به شکل زیر هستند، تعداد 640 بیمار که به ازای هر کدام 2 ویژگی مرتبط به ژن ها داریم و میخواهیم بینیم با توجه به تفاوت این ویژگی ها در افراد، آیا میشود آن ها را در خوش های مجزا قرار داد؟

```
data=[[15.749946, 25.67278], [18.836815, 25.197464], [-7.8372335, 10.255067], [-26.534391, -2
label=['KIRC', 'KIRC', 'BRCA', 'PRAD', 'BRCA', 'BRCA', 'BRCA', 'LUAD', 'KIRC', 'PRAD', 'BRCA'
```

در زیر نمونه ها را بدون برچسب هایشان رسم میکنیم و به طور شهودی میتوان دید داده ها به 5 خوش های مجزا قابل تفکیک اند.

```
import numpy as np
import matplotlib.pyplot as plt

# plotting data wrt. its features
X=np.array(data)
A=X.T[0]
B=X.T[1]
plt.scatter(A,B)
plt.show()
```



سپس یک مدل kmeans با تعداد 5 خوش را روی این داده ها آموزش میدهیم و اندیس خوش ای که هر نمونه به آن منتب شده را نمایش میدهیم.

توجه کنید که اندیس خوش را با برچسب داده اشتباه نگیرید. ما روی داده ها برچسب نزده ایم که هر کدام به چه نوع سلطانی تعلق دارند. صرفا نمونه ها را خوش بندی کردیم و گفتیم هر کدام متعلق به کدام خوش هستند.

```
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters=5).fit(data)
```

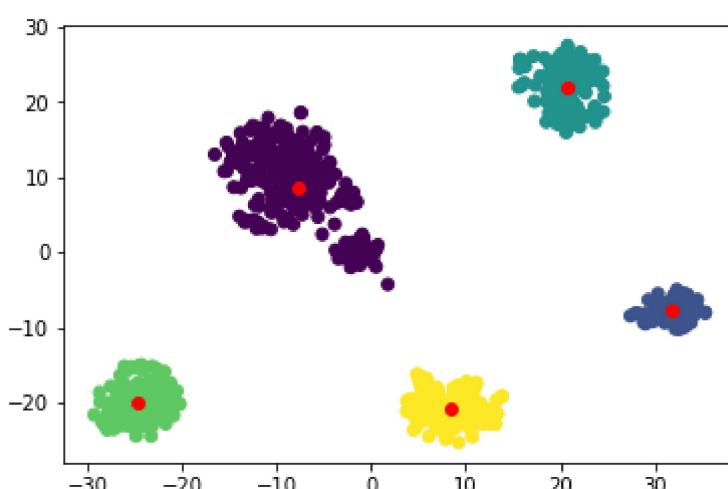
```
y_pred=kmeans.predict(data)
```

```
y_pred
```

```
array([1, 1, 2, 0, 2, 2, 2, 3, 1, 0, 2, 0, 4, 3, 4, 2, 0, 0, 1, 2, 2, 4,
       1, 1, 2, 2, 2, 2, 2, 1, 0, 2, 2, 3, 0, 2, 1, 1, 2, 0, 2, 2,
       4, 4, 2, 2, 2, 2, 1, 3, 2, 1, 2, 0, 4, 2, 0, 2, 4, 3, 2, 3, 3, 4,
       2, 4, 2, 1, 3, 0, 4, 0, 1, 0, 1, 4, 2, 0, 3, 2, 1, 2, 3, 3, 3, 2,
       2, 1, 2, 0, 2, 2, 2, 1, 0, 2, 0, 3, 2, 2, 2, 2, 1, 3, 2, 2, 2,
       3, 2, 2, 1, 1, 2, 3, 2, 1, 1, 3, 2, 3, 3, 3, 0, 2, 1, 0, 4, 1,
       2, 3, 1, 1, 2, 2, 3, 2, 2, 0, 0, 2, 4, 3, 3, 1, 1, 0, 2, 3, 1, 2,
       3, 1, 4, 2, 3, 2, 3, 0, 1, 4, 2, 2, 4, 0, 1, 1, 2, 2, 1, 0, 0, 3,
       3, 0, 4, 2, 0, 1, 2, 1, 2, 4, 3, 2, 3, 1, 4, 1, 2, 0, 0, 1, 3,
       4, 1, 3, 4, 3, 2, 3, 4, 0, 2, 2, 3, 1, 3, 2, 1, 2, 4, 3, 2, 3, 2,
       0, 0, 1, 2, 1, 3, 0, 0, 2, 3, 4, 2, 2, 1, 3, 3, 2, 1, 2, 4, 3, 0,
       2, 1, 4, 3, 3, 4, 0, 2, 2, 1, 0, 2, 1, 2, 1, 2, 3, 3, 2, 0, 3, 4,
       1, 2, 4, 2, 3, 4, 2, 4, 0, 4, 4, 2, 2, 4, 3, 2, 3, 0, 4, 0, 2, 2,
       2, 3, 2, 1, 2, 1, 0, 2, 2, 4, 2, 0, 2, 0, 3, 1, 2, 0, 0, 2, 4,
       2, 3, 4, 3, 2, 0, 0, 2, 0, 3, 3, 0, 3, 4, 2, 0, 2, 2, 4, 2, 2, 4,
       0, 2, 2, 3, 2, 1, 2, 2, 0, 0, 0, 4, 3, 1, 1, 2, 1, 3, 2, 4, 2, 3,
       2, 2, 2, 3, 2, 2, 1, 1, 0, 3, 2, 2, 1, 2, 1, 0, 3, 3, 2, 2, 0,
       2, 1, 2, 3, 2, 0, 2, 1, 2, 4, 3, 0, 1, 1, 1, 2, 2, 3, 2, 3, 2, 0,
       1, 2, 2, 3, 1, 3, 2, 3, 2, 1, 0, 1, 0, 1, 3, 3, 1, 0, 0, 3, 1, 0,
       1, 4, 2, 1, 1, 2, 1, 1, 3, 2, 0, 3, 2, 3, 2, 1, 0, 2, 3, 1, 2, 2,
       4, 2, 2, 1, 0, 1, 2, 2, 2, 1, 0, 1, 1, 2, 2, 3, 2, 4, 0, 1, 1, 1,
       2, 1, 0, 0, 2, 1, 2, 1, 3, 2, 1, 2, 1, 4, 3, 0, 3, 2, 2, 2, 3, 2,
       4, 4, 2, 2, 2, 0, 2, 4, 0, 2, 2, 1, 1, 2, 2, 2, 2, 3, 4, 4, 0,
       4, 1, 3, 0, 1, 2, 1, 2, 2, 1, 1, 2, 1, 1, 0, 0, 3, 3, 3, 2, 2, 2,
       0, 0, 4, 2, 3, 1, 2, 3, 4, 1, 2, 1, 0, 4, 2, 0, 0, 2, 2, 0, 2, 2,
       2, 3, 2, 2, 3, 0, 0, 1, 4, 2, 2, 4, 2, 1, 0, 2, 0, 2, 2, 0, 2, 1,
       2, 1, 4, 4, 0, 2, 2, 0, 2, 1, 0, 0, 2, 3, 0, 3, 1, 2, 0, 2, 4, 3,
       3, 3, 0, 3, 2, 2, 3, 0, 1, 1, 2, 3, 3, 0, 2, 0, 4, 0, 4, 1, 2, 0,
       2, 1, 2, 2, 2, 0, 2, 3, 1, 0, 1, 1, 2, 0, 2, 2, 0, 2, 0, 3, 3, 2,
       1, 0])
```

نمونه ها را با توجه به خوش هایشان رنگ آمیزی میکنیم، مراکز خوش نیز در نمودار نشان داده شده اند.

```
A=np.array(data).T[0]
B= np.array(data).T[1]
plt.scatter(A,B, c =y_pred)
plt.scatter( kmeans.cluster_centers_.T[0], kmeans.cluster_centers_.T[1],color="red")
plt.show()
```



مراکز خوشه ها از طریق `_kmeans.cluster_centers_` قابل دسترسی هستند.

```
kmeans.cluster_centers_
```

```
array([[ -7.71792019,    8.58312359],
       [ 31.69665558,   -7.66848647],
       [ 20.6374405 ,   22.03263534],
       [-24.59613826,  -19.93664994],
       [  8.35913904,  -20.8285965 ]])
```

هم چنین میتوانیم برای داده های جدید مشخص کنیم به کدام خوشه متعلق اند.

```
X_new = np.array([[0, 2], [15, 25], [-30, -15], [10,-10]])
kmeans.predict(X_new)
```

```
array([0, 2, 3, 4])
```

نکته : به جای اینکه به طور سخت تعیین کنیم هر نمونه متعلق به کدام خوشه است، میتوان فاصله‌ی آن‌ها با مراکز خوشه‌ها را نمایش داد به نوعی هرچقدر فاصله‌ی نمونه با مرکز خوشه‌ای کمتر از فاصله‌اش با سایر مرکز خوشه‌ها باشد، احتمال قرار گرفتن در آن خوشه نسبت به دیگر خوشه‌ها بیشتر است و این یک رویکرد برای خوشبندی نرم خواهد بود.

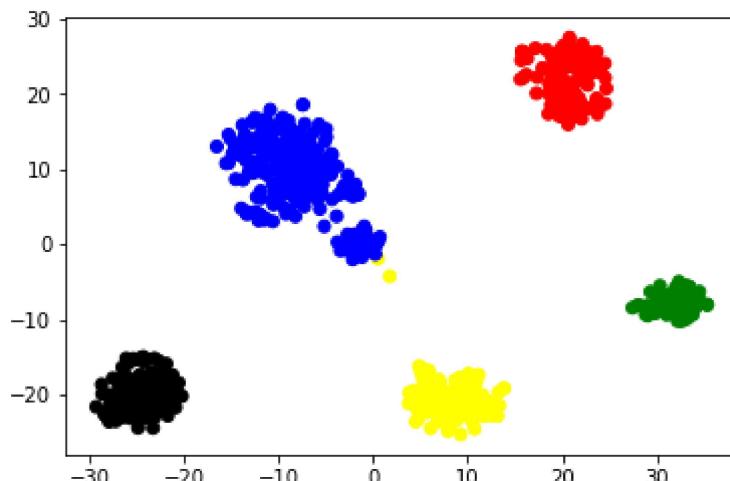
در ماتریس فاصله‌های زیر، میتوان هر سطر را به مجموع آن سطر تقسیم کرد و احتمالاتی بین ۰ و ۱ نیز داشت.

```
kmeans.transform(X_new)
```

```
array([[10.14415143, 33.13846112, 28.76126613, 32.95734558, 24.31090339],
       [28.02887313, 36.68798599, 6.37071334, 59.89287664, 46.3072488 ],
       [32.44464207, 62.13073636, 62.73409329, 7.31930559, 38.79943408],
       [25.67600394, 21.82156774, 33.75270163, 35.99485789, 10.95221105]])
```

چون در مثال ما برچسب‌ها را داشتیم دست به یک آزمایش میزنیم، در نهایت برچسب واقعی داده‌ها را نمایش میدهیم تا به یک شهود برسیم که واقعاً هر خوشه‌ی یافت شده یک گونه‌ی سرطان متفاوت بوده است و میتوان نتیجه گرفت در کاربردهای واقعی هر خوشه میتواند تفاوت معناداری با سایر خوشه‌ها داشته باشد.

```
col={"BRCA":"blue", "KIRC":"red", "COAD":"green", "LUAD":"yellow" , "PRAD":"black"}
YC=[]
for i in range(len(label)):
    YC.append(col[label[i]])
plt.scatter(A,B, c =YC)
plt.show()
```



3-2. مقداردهی اولیه‌ی مرکز خوش‌ها

یک چالش پیش رو در استفاده از این الگوریتم این است که هرچند الگوریتم تضمین می‌کند همگرا شود، نمیتواند تضمین کند که به بهترین پاسخ (بهینه‌ی سراسری) همگرا می‌شود و پاسخ الگوریتم می‌تواند بسته به مقدار دهی اولیه‌ی مرکز خوش‌ها می‌تواند متفاوت باشد و به بهینه‌های محلی همگرا شود.

برای روبرویی با این چالش چند روش برای مقداردهی اولیه‌ی مرکز خوش‌ها پیشنهاد شده است که در ادامه مورد بررسی قرار می‌دهیم.

الف) اگر به طریقی یک تقریب از مرکز خوش‌ها داشته باشیم که از حالت رندوم بهتر باشد (مثلاً قبل از الگوریتم خوشبندی دیگر را اجرا کرده باشیم) می‌توانیم به طور دستی آن را به جای مقادیر رندوم، به عنوان مقادیر اولیه‌ی مرکز خوش‌ها ست کنیم. به این منظور می‌توان مرکز خوش‌های پیشنهادی را در یک `np.array` ذخیره کرد و با استفاده از هایپرپارامتر `init` مقدار اولیه‌ی مرکز خوش‌ها را ست کرد.

```
good_init = np.array([[8, -20], [-8, 7], [19, 25], [-25, -20], [30, -10]])  
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1).fit(data)
```

ب) راه حل دیگر این است که الگوریتم‌ها به تعداد دفعات معین اجرا کنیم و از میان آن‌ها بهترین پاسخ را انتخاب کنیم. واضح است برای انجام این کار نیاز داریم معیاری برای مقایسه‌ی عملکرد داشته باشیم (`performance metric`) تا بتوانیم بهترین آن‌ها را انتخاب کنیم. به این معیار `model interia` گفته می‌شود که عبارت است از میانگین مجدورات فاصله بین داده‌ها و مرکز خوش‌های ایشان است که طبیعتاً هرچقدر این مقدار کمتر باشد یعنی مدل بهتر بوده. به این منظور می‌توان هایپرپارامتر `n_init` را به تعداد دفعاتی که می‌خواهیم الگوریتم اجرا شود ست کنیم.

توجه: اگر این هایپرپارامتر را ست نکنیم مقدار پیش فرض آن 10 خواهد بود.

نکته: با استفاده از `kmeans.inertia_` می‌توان به مدل نهایی دست پیدا کرد. هم چنین `score` برای داده ای که خوشبندی روی آن انجام شده همواره برابر با `-inertia` خواهد بود. زیرا `score` طبق تعریف باید اینگونه باشد که بیشتر بودنش به معنای بهتر بودنش باشد.

```
kmeans = KMeans(n_clusters=5, n_init=20).fit(data)  
print("Interia:", kmeans.inertia_, "Score:", kmeans.score(X))
```

Interia: 14489.701572428849 Score: -14489.701572428849

4-2. بهبودهای K-means

++K-means.2-4-1

یکی از بهترین بهبودهای پیشنهاد شده برای الگوریتم `kmeans` است که با داشتن یک روش هوشمندانه‌تر برای مقدار دهی اولیه‌ی مرکز خوش‌های ای انتخاب شوند که از یکدیگر دور باشند. نشان داده شده اجرای این ایده‌ی کلی این است که مرکز خوش‌های ای انتخاب شوند که از یکدیگر دور باشند. نشان داده شده اجرای این بخش به طرز شایان توجهی تعداد دفعات اجرای الگوریتم برای رسیدن به جواب بهینه را کم می‌کند و از همین جهت محاسبات اضافه‌ی لازم برای این مقداردهی اولیه، ارزشمند است.

الگوریتم به شکل زیر عمل می‌کند:

یک نمونه‌ی رندوم را به عنوان مرکز خوش‌های اول $(^{(1)}c)$ انتخاب کن

تا زمانی که تمام k مرکز خوش انتخاب شوند : نمونه‌ی $x^{(i)}$ را با احتمال $D(x^{(i)})^2 / \sum_{j=1}^m D(x^{(i)})^2$ به عنوان مرکز خوش‌ی بعدی $c^{(i)}$ انتخاب کن. که $c^{(i)}$ با نزدیک‌ترین مرکز خوش‌ی تا به حال انتخاب شده است.

مشخص است که سیاست در انتخاب مرکز خوش‌ها به گونه‌ای است که نمونه‌هایی که از مراکز خوش‌های فعلی انتخاب شده فاصله‌ی بیشتری را دارند، به مراتب احتمال بیشتری برای انتخاب شدن دارند.

K-means استفاده شده در کدهای بالا به طور پیش فرض از همین روش برای مقدار دهی اولیه استفاده می‌کند، در صورت نیاز به مقدار دهی اولیه‌ی رندم باید هایپرپارامتر `init` را برابر با `random` ست کنیم.

2-4-2 K-means تسريع شده

این الگوریتم با تلاش برای عدم انجام محاسبات غیر ضروری تا حد قابل توجهی الگوریتم اصلی را تسريع می‌کند. رویکرد کلی استفاده از نامساوی مثلثی و در نظر گرفتن کران‌های بالا و پایین برای فاصله‌ی نمونه‌ها و مراکز خوش است.

استفاده شده در کدهای بالا به طور پیش فرض از همین روش استفاده می‌کند، در صورت نیاز به اجرای الگوریتم به روش اصلی، باید هایپرپارامتر `algorithm` را برابر با `full` ست کنیم.

2-4-3 mini-batch K-means

این الگوریتم k را قادر می‌سازد به جای استفاده از تمام نمونه‌ها در هر `iteration`، فقط از یک دسته یا `mini batch` از داده‌ها برای آپدیت کردن و جابجا کردن مراکز خوش استفاده کند که در حدود 3 تا 4 بار الگوریتم را تسريع می‌کند. هم چنین استفاده از `Mini batch` این مزیت را دارد که الگوریتم برای داده‌های عظیم که برای جا شدن تمام نمونه‌هایشان در مموری مشکل دارند نیز قابل اجرا خواهد بود. به این منظور به شکل زیر عمل می‌کنیم.

```
from sklearn.cluster import MiniBatchKMeans
minibatch_kmeans = MiniBatchKMeans(n_clusters=5)
minibatch_kmeans.fit(data)

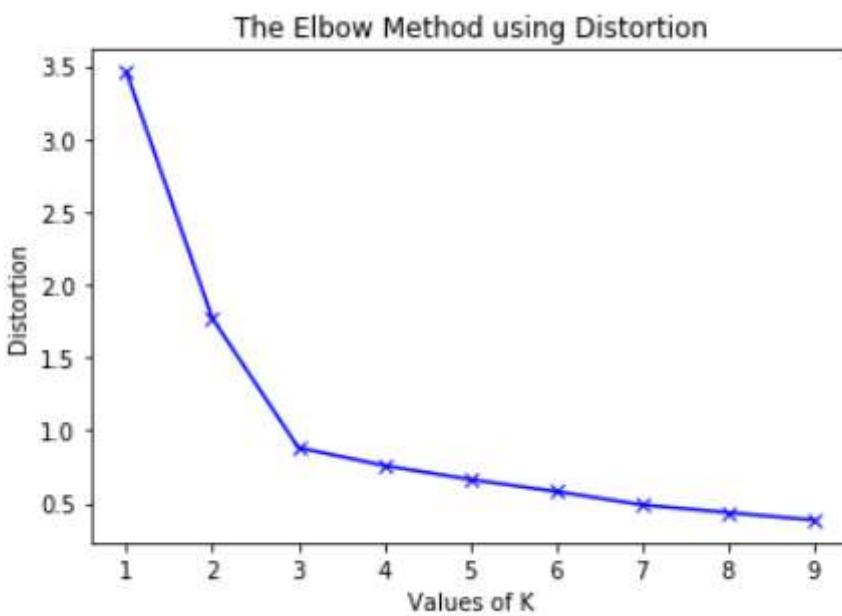
MiniBatchKMeans(n_clusters=5)
```

هرچند، استفاده از `mini batch` باعث افزایش سرعت بسیار زیادی می‌شود؛ اما در اکثر موقع `inertia` به مقدار کمی نسبت به حالت عادی بیشتر خواهد بود. این تفاوت خصوصاً با افزایش تعداد خوش‌ها نمایان‌تر می‌شود.

5-2. انتخاب تعداد خوش‌ها

تا اینجا ما فرض کردیم تعداد خوش‌ها را میدانیم و در مثال‌های فوق تعداد خوش‌ها از نگاه کردن به نمونه‌ها قابل تشخیص بود؛ اما برای کاربردهای واقعی این موضوع صادق نخواهد بود. اگر k بیش از حد زیاد یا کم انتخاب شود خوش‌بندی مناسبی را نخواهیم داشت پس مهم است روشی برای یافتن مقدار مناسب k داشته باشیم.

5-2-1 استفاده از `inertia`



[pic source](#)

ممکن است در ابتدا اینگونه به نظر برسد که k ای که منجر به مدلی با inertia کمتر شود، k مناسب تری خواهد بود؛ اما متأسفانه این موضوع درست نیست. هرچقدر k یا همان تعداد خوشها افزایش یابد، Inertia کاهش میابد؛ زیرا تعداً مرکز خوش بیشتر خواهد بود و فاصله‌ی هر نمونه با نزدیک‌ترین مرکز خوش اش کمتر خواهد شد تا حدی که اگر به اندازه‌ی تعداد نمونه‌ها خوش داشته باشیم هر نمونه مرکز خوشی خود خواهد بود و $\text{Inertia} = 0$ را خواهیم داشت. پس کمتر بودن Inertia به معنای k مناسب‌تر نیست.

به نمودار بالا دقت کنید علاوه بر اینکه با افزایش تعداد خوشها Inertia کاهش میابد یک موضوع دیگر را نیز می‌توان از نمودار برداشت کرد. از مقدار $k=1$ to 4 مقدار inertia با شدت چشمگیری در حال کاهش است در حالی که بعد از 4 کاهش بسیار آهسته‌تر می‌شود. این بدان معناست که کاهش inertia از 4 به بعد تنها به دلیل افزایش تعداد خوش هاست اما از قبل از آن این تفاوت معنی دار بوده است و در واقع افزایش تعداد خوشها در آن مراحل gain زیادی برای ما دارد. در واقع نمودار به شکل یک دست خواهد بود که قسمت شکستگی نمودار که شبیه آرنج است (یعنی جایی که شدت کاهش کم می‌شود) می‌تواند انتخاب خوبی برای تعداد خوشها باشد.

silhouette score.2-5-2

یک روش دقیق‌تر و البته با محاسبات بیشتر برای تعیین بهترین تعداد خوش، استفاده از silhouette score است. silhouette score برابر است با میانگین silhouette coefficient روی تمام نمونه‌ها.

$$\text{Silhouette-coefficient} = (b - a) / \max(a, b)$$

که در آن a میانگین فاصله‌ی نمونه با تمام نمونه‌های هم خوش اش است (میانگین فاصله‌ی درون خوش‌ای) و b میانگین فاصله‌ی نمونه با نمونه‌های نزدیک‌ترین خوش است (درواقع b کمترین میانگین فاصله با نمونه‌های خوش‌های دیگر است).

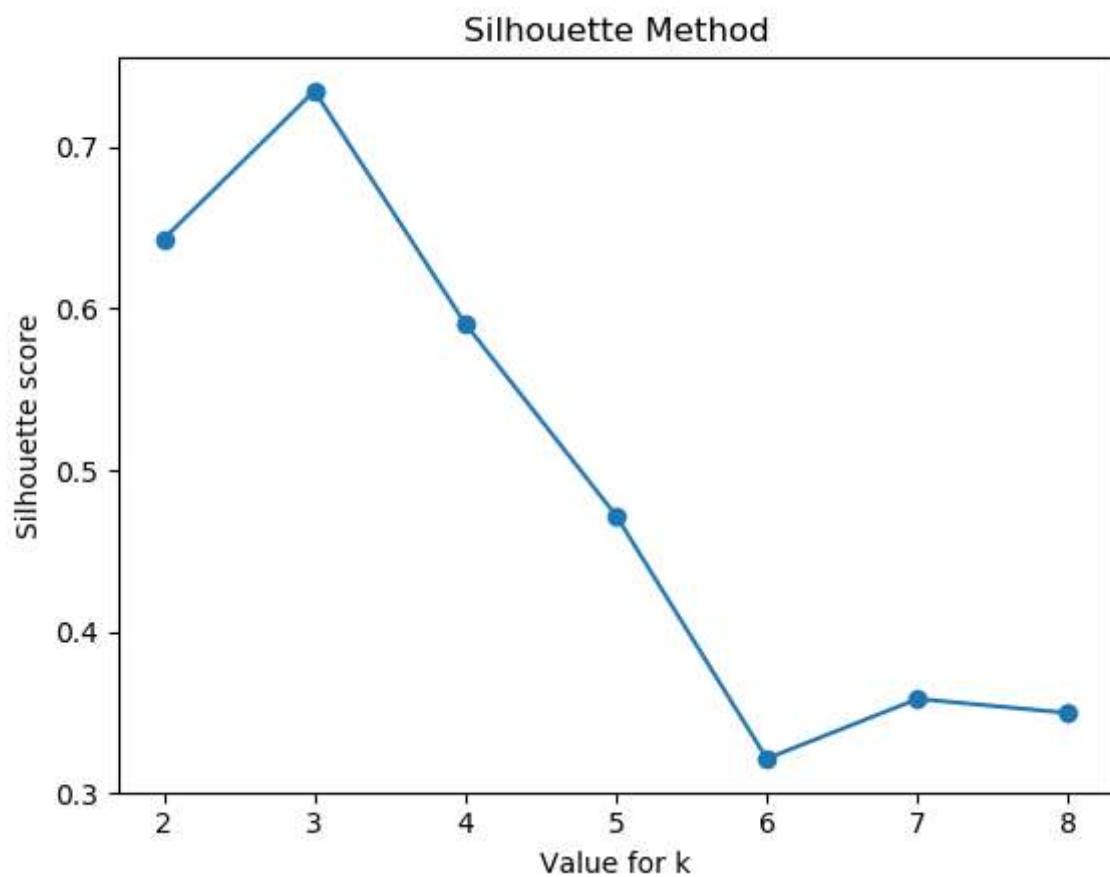
مقدار silhouette coefficient می‌تواند بین -1 و $+1$ باشد که مقدار نزدیک به $+1$ نشان می‌دهد نمونه به نمونه‌های درون خوشی نسبت یافته اش نزدیک و از نمونه‌های سایر خوشها دور است که در واقع یعنی به درستی خوش اش انتخاب شده و مقدار نزدیک به -1 می‌تواند بیانگر این باشد که نمونه به خوش‌ی اشتباہی نسبت داده شده است.

به شکل زیر می‌توان silhouette score برای داده‌ها را محاسبه کرد.

```
from sklearn.metrics import silhouette_score
```

```
silhouette_score(data, kmeans.labels_)
```

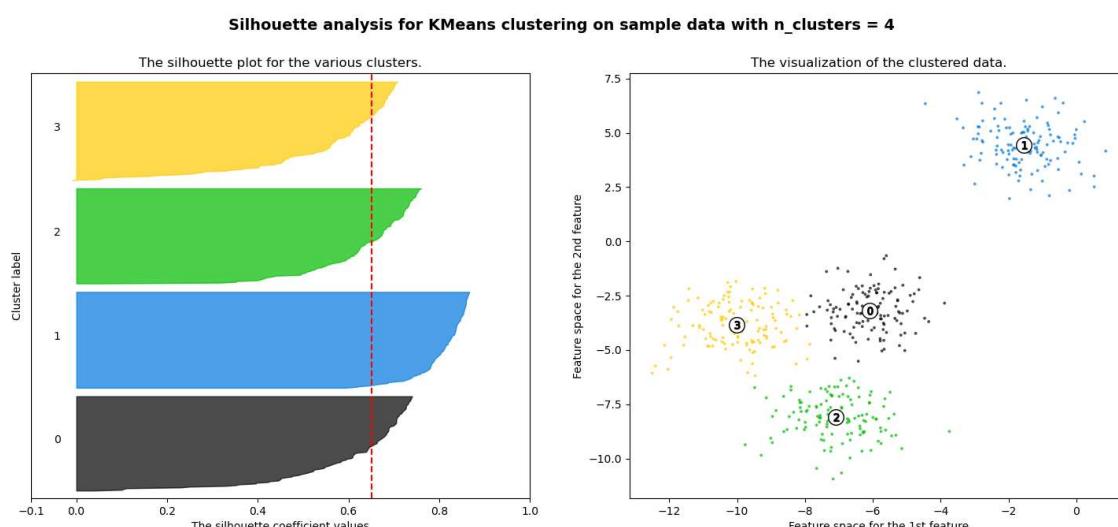
```
0.8062882462874054
```



[pic source](#)

به این ترتیب می‌توانیم Silhouette score را برای k های متفاوت محاسبه و مقایسه کنیم و بیشتر به معنای k مناسب‌تر خواهد بود. هم چنین استفاده از Silhouette score برخلاف inertia به ما این امکان را می‌دهد که دو k ای دلخواه را نسبت به هم مقایسه کنیم (در روش قبل فقط k بهینه را با استفاده از محل شکستگی پیدا می‌کردیم).

علاوه بر نمایش silhouette score به ازای k های مختلف می‌توان از silhouette diagram نیز استفاده کرده که در آن silhouette coefficient برای تمام نمونه‌ها به ترتیب شماره‌ی خوش‌شان و سپس ترتیب بزرگی silhouette score نمایش داده می‌شود. خط عمودی در این نمودار نیز silhouette coefficient به ازای k ای است که خوشبندی با ان انجام شده.



[pic source](#)

اگر silhouette coefficient خوش‌هایی با خط عمودی فاصله داشته باشند نشان دهنده‌ی این است که خوش‌بندی بد انجام شده و خوش‌های دیگر خوش‌ها فاصله‌ی کافی ندارند.

نکته: از این نمودار می‌شود اندازه‌ی نسبی خوش‌ها نسبت به یکدیگر را نیز دید. در شرایطی که برای دو k شرایط تقریباً یکسان بود یک انتخاب خوب می‌تواند انتخاب k ای باشد که با استفاده از آن اندازه‌ی خوش‌ها در آن هم

.....

DB index.2-5-3

یک روش دیگه برای سنجیدن کیفیت خوش‌بندی انجام شده Davies-Bouldin index است. تعاریف زیر را در نظر بگیرید:

پراکندگی خوش: می‌تواند به صورت یک انحراف معیار تعمیم یافته تعبیر شود.

$$\delta_k := \sqrt{\frac{1}{N_k} \sum_{n \in n_k} \|x_n - c_k\|^2}$$

شماحت خوش: دو خوش‌های شبیه در نظر گرفته می‌شوند اگر نسبت به فاصله شان پراکندگی زیادی داشته باشند.

$$S_{kl} := \frac{\delta_k + \delta_l}{\|c_k - c_l\|}$$

با توجه به تعاریف بالا می‌توان دید DB-index که به صورت زیر تعریف می‌شود می‌تواند معیار خوبی برای سنجش کیفیت خوش‌بندی باشد.

$$V_{DB} := \frac{1}{k} \sum_{k=1}^K \max_{l \neq k} S_{kl}$$

2. محدودیت‌های k-means

علی‌رغم ویژگی‌های مثبت kmeans مانند سریع و مقیاس پذیر بودنش، این الگوریتم محدودیت‌های قابل توجهی دارد از جمله:

- وابستگی به مقدار اولیه و نیاز به چندین بار اجرا با مقادیر اولیه متفاوت
- نیاز به مشخص کردن تعداد خوش‌ها
- مشکل در خوش‌بندی نمونه‌هایی که خوش‌هایشان هم اندازه نیستند
- مشکل در خوش‌بندی نمونه‌هایی که چگالی خوش‌ها در آن متفاوت است
- مشکل در خوش‌بندی نمونه‌هایی که خوش‌ها فرم غیر کروی (غیر دایره‌ای) دارند

نکته: برای مشکل آخر، می‌توان با استفاده از اسکیل کردن ویژگی‌ها، فرم خوش‌ها را به فرم کروی نزدیک‌تر کرد. هرچند تضمین صد درصد وجود نداره که اسکیل کردن ویژگی‌ها تمام خوش‌ها را آل کند؛ اما معمولاً باعث بهبود عملکرد می‌شود.

3. کاربردهای خوش‌بندی

Image Segmentation .3-1

هدف کلی Image segmentation این است که یک تصویر به بخش‌هایی تقسیم شود که هر کدام از این بخش‌ها یک موجودیت مجزای مورد نظر باشد. مثال: تشخیص موانع در اتوموبیل‌های خودران

هرچند مدل‌های Image segmentation برای مثال‌هایی مانند اتوموبیل خودران می‌توانند بسیار پیچیده باشد و به عنوان مثال نیاز به شبکه‌های عصبی و ... باشد، در موارد ساده‌تر می‌توان رویکرد ساده‌تری به نام Color segmentation را استفاده کرد. در این رویکرد ما بخش‌هایی از تصویر را که رنگ تقریباً مشابهی دارند به عنوان یک گروه مجزا (یک خوش) جدا می‌کنیم و پیش فرض می‌کنیم که قسمت‌های همنگ تصویر احتمالاً متعلق به یک موجودیت هستند.

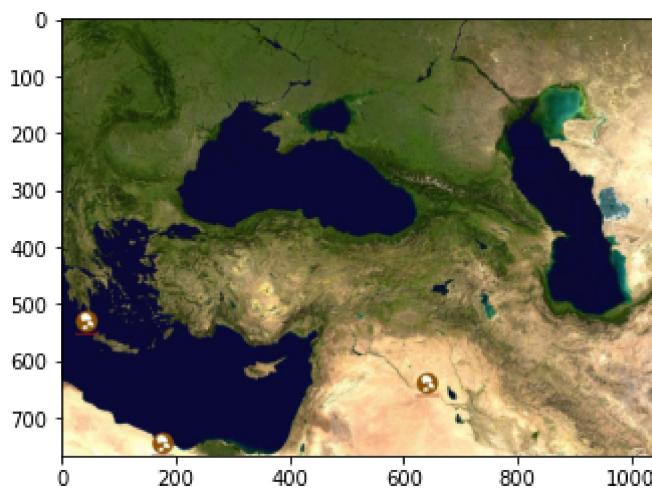
این رویکرد ساده در برخی مثال‌ها می‌تواند راه‌گشا باشد برای مثال تشخیص اینکه چند درصد از یک تصویر ماهواره‌ای از زمین، جنگل و چند درصد اقیانوس است.

بیایید مثال تشخیص درصد اقیانوس و جنگل را اجرا کنیم. در مثال زیر ما یک تصویر ماهواره‌ای داریم و با استفاده از خوش بندی در واقع هر پیکسل از تصویر را به یک خوش احتصاص میدهیم که از نظر رنگ کمترین فاصله را با مرکز خوش اش دارد و سپس تمام اعضای هر خوش را با میانگین مقدار خوش جایگزین می‌کنیم.
لود کردن تصویر و نمایش :

```
from matplotlib.image import imread
import matplotlib.pyplot as plt
import numpy as np

image = imread("./map.jpg")

plt.imshow((image).astype(np.uint8))
plt.show()
```



[pic source](#)

خوش بندی تصویر با $k=3$ (با دانش قبلی که تصویر ماهواره‌ای اصولاً از سه بخش خشکی آب و جنگل تشکیل شده) و سپس جاگذاری هر پیکسل با میانگین مقدار خوش اش:

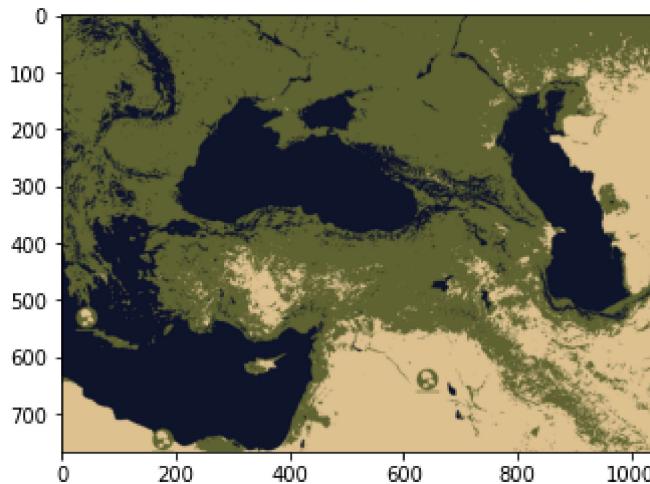
```
X = image.reshape(-1, 3)
kmeans = KMeans(n_clusters=3).fit(X)
segmented_img = kmeans.cluster_centers_[kmeans.labels_]
segmented_img = segmented_img.reshape(image.shape)
colors=kmeans.labels_
```

نمایش تصویر پس از خوش بندی می‌توان دید به خوبی جنگل‌ها در یک خوش، آب‌ها در یک خوش و خشکی‌ها در یک خوش قرار گرفتند.

```

out = segmented_img
plt.imshow((out).astype(np.uint8))
plt.show()

```



نمایش درصد هر خوشه :

```

l=list(colors)
print("Cluster0:",l.count(0)/len(l),"%\nCluster1:",l.count(1)/len(l),"%\nCluster2:",l.count(2)

Cluster0: 0.293452066655489 %
Cluster1: 0.47722637491335235 %
Cluster2: 0.22932155843115862 %

```

نکته: همانطور که گفته شد ما تعداد خوشه هارا با استفاده از کاربرد خود و دانش قبلی انتخاب کردیم. در مثال های دیگر نیز باید با بررسی های قبلی تعداد خوشه ها را انتخاب کنیم. اگر تعداد خوشه ها کم باشد دو موجودیت مجزا در تصویر که میخواستیم تفکیکشان کنیم در یک دسته قرار خواهند گرفت مثلاً اگر تعداد خوشه ها در بالا 2 باشد جنگل و آب ها در یک خوشه ادغام شده و نمیتوانیم درصد آن ها را مشخص کنیم. مشخص است که اگر خوشه ها بیش از حد زیاد انتخاب شوند نیز مطلوب نیست.

2-3. پیش پردازش دادهها

یکی از پیش پردازش های مهم کاهش ابعاد است. ما با کاهش ابعاد سعی می کنیم ابعاد داده های خود را کمتر کنیم به گونه ای که داده در ابعاد کمتر همچنان اطلاعات کافی برای بیان داده ها را دارا باشد به همین علت اصولاً داده ها در ابعاد پایین تر، اطلاعات نامفید و اضافی که ممکن است داده ای اولیه داشته باشد را نخواهند داشت و البته طبیعتاً بخشی از اطلاعات اصلی داده را از دست میدهند که اگر متدهای خوب نباشد آن بخش ها میتوانند بخش های مفیدی بوده باشند.

یک روش کاهش ابعاد استفاده از خوشه بندی است به این شکل که ما داده ها را خوشه بندی کنیم و هر داده را با وکتور فاصله اش از تمام مرکز داده ها جایگزین کنیم (در مثال های بالا دیدیم که میتوان فاصله ای هر نمونه با مرکز خوشه را پس از خوشه بندی داشت). به این شکل ابعاد داده ای خود را از بعد اولیه n به k که تعداد خوشه هاست کاهش دادیم ($n > k$) و میدانیم این بیان جدید داده معنا دار است؛ چون هر خوشه داده های شبیه به هم را در خود دارد و فاصله از هر خوشه بیانگر شباهت یا تفاوت نمونه نسبت به نمونه های سایر خوشه هاست و در واقع این بیان جدید از داده ها، دور یا نزدیک بودن آن ها نسبت به هم و ارتباطشان نسبت به یکدیگر را در خود حفظ کرده است.

مزیت کاهش ابعاد داده ها این است که اولاً حجم داده ها کم می شود و منابع لازم برای محاسبات و ذخیره سازی داده ها کمتر خواهد بود و دوماً اینکه اگر از روش خوبی برای کاهش ابعاد استفاده شده باشد همانطور که گفته شد اطلاعات غیرضروری در داده های کاهش بعد داده شده نخواهند بود و وقتی مدل برآموزش داده های با کیفیت تری

در اختیار داشته باشد طبیعتاً عملکرد بهتری خواهد داشت.

بعد از کاهش ابعاد دادهها با استفاده از خوشه‌بندی می‌توان از آن‌ها برای آموزش یک مدل به طور با ناظر (با فرض موجود بودن برچسب داده‌ها) استفاده کرد.

در مثال‌های زیر از دیتابیس ارقام دست نویس از دیتاست‌های موجود در sklearn.datasets استفاده شده است که در آن هر نمونه یک تصویر 8^*8 از یک رقم دست نویس است.
لود کردن دیتابیس و نمایش یکی از نمونه‌های آن :

```
from sklearn.datasets import load_digits
```

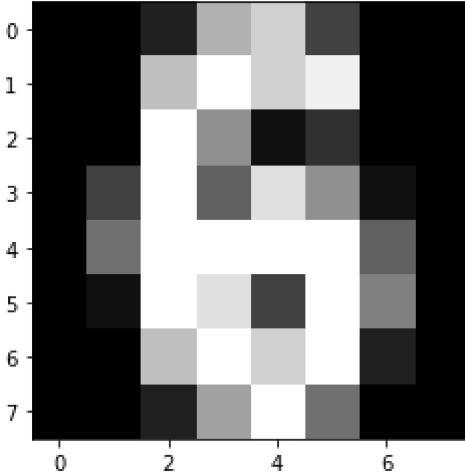
```
X_digits, y_digits = load_digits(return_X_y=True)
```

```
X_digits.shape
```

```
(1797, 64)
```

```
i=402
```

```
X=X_digits[i].reshape(8,8)  
plt.imshow(X,cmap='gray')  
plt.show()
```



همانطور که در بالا دیده شد هر نمونه یک تصویر 8^*8 است یعنی هر نمونه 64 پیکسل یا ویژگی دارد. (در داده نیز این موضوع مشخص است)

حال با kmeans داده‌ها را با $k=20$ خوشه‌بندی می‌کنیم و به ازای هر نمونه به جای خود نمونه، فاصله اش از مرکز خوشه‌ها را جایگزین می‌کنیم.

در واقع هر نمونه ی 64 پیکسلی با یک آرایه 20 عنصری جایگزین می‌شود یعنی ابعاد ویژگی‌ها را از 64 به 20 کاهش داده ایم.

توجه: هرچند شاید به نظر برسد تعداد خوشه‌ها باید 10 باشد و 10 خوشه برای بیان داده‌ها کافی است، این موضوع را مد نظر داشته باشید که هر رقم به شکل‌های متفاوتی نوشته می‌شود پس ممکن است با 10 خوشه به نتیجه خوبی نرسیم.

```
kmeans = KMeans(n_clusters=20).fit(X_digits)  
kmeans.transform(X_digits).shape
```

```
(1797, 20)
```

بخشی از داده های کاهش بعد داده شده را به عنوان تست و بخش دیگر را به عنوان داده های آموزش جدا میکنیم و یک مدل logistic regression روی آن آموزش میدهیم. میتوان دید علی رغم اینکه داده ها را کاهش بعد داده بودیم به دقت خوبی رسیدیم. کاهش بعد طبیعتا سرعت آموزش را بالا میبرد زیرا مدل پارامتر های کمتری خواهد داشت.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(kmeans.transform(X_digits), y_digits)
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)
log_reg.score(X_test, y_test)

0.9755555555555555
```

Semi-Supervised learning .3-3

وقتی تعداد زیادی دادهی بدون برچسب داریم میتوانیم از روش های یادگیری بدون ناظر استفاده کنیم. اما برای تسک هایی مانند کلاس بندی نیاز داریم داده ها برچسب داشته باشند و از آنجا که نمیتوانیم تمام داده ها را برچسب بزنیم (به دلیل هزینه بر بودن) مجبوریم تعداد کمی از آن ها را به نحوی انتخاب کنیم و فقط آن ها را برچسب زده و برای آموزش استفاده کنیم. میتوان حدس زد اگر به طور کاملاً رندوم بخش نسبتا کمی از داده ها را برچسب بزنیم و از آنها برای آموزش مدل استفاده کنیم نتیجه هی مطلوبی نخواهیم داشت. (در مثال زیر 50 نمونه از کل داده های ارقام دست نویس)

توجه : منظور از برچسب گذاری باید این باشد که ما پس از انتخاب نمونه ها به طور دستی تک به تک آن ها را ببینیم و برچسب آن ها را به طور دستی مشخص کنیم اما چون در مثال ها در ادامه برچسب همه داده ها از قبل موجود است ما صرفا برچسب نظیر داده های انتخابی را به عنوان برچسبشان برای آموزش استفاده میکنیم. توجه کنید در یک مثال semi-supervised دنیای واقعی برچسب ها را در ابتدا نداریم و باید خودمان برچسب داده های انتخاب شده را مشخص کنیم.

```
from sklearn.datasets import load_digits
X_digits, y_digits = load_digits(return_X_y=True)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_digits, y_digits)
n_labeled = 50
log_reg = LogisticRegression()
log_reg.fit(X_train[:n_labeled], y_train[:n_labeled])
log_reg.score(X_test, y_test)

0.7711111111111111
```

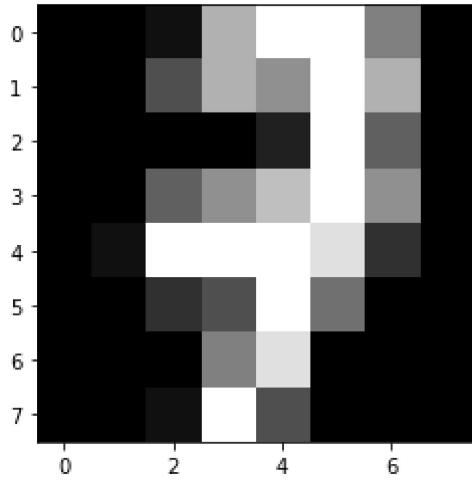
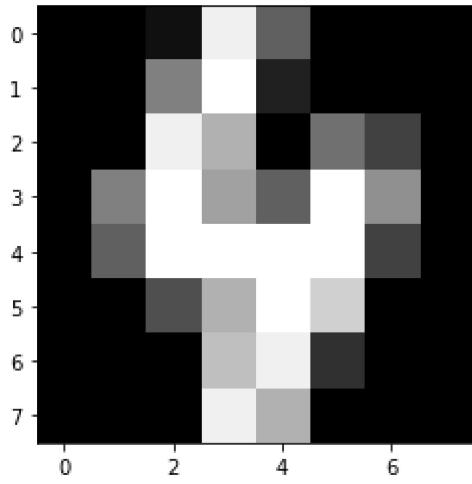
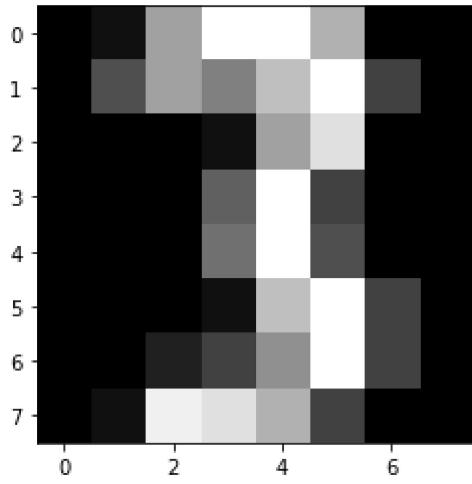
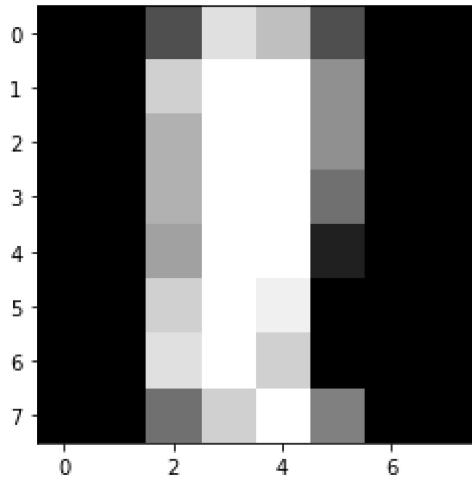
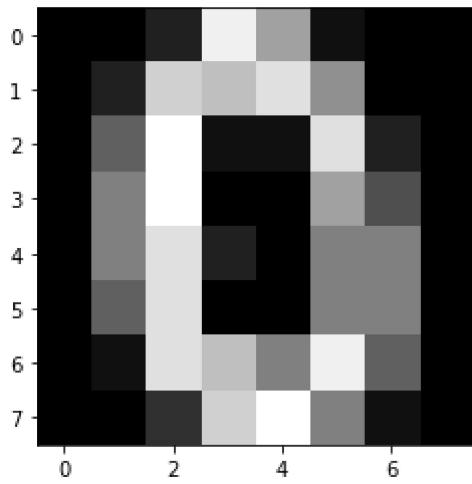
اینجا این ایده مطرح می شود که با استفاده از یک روش بدون ناظر (در اینجا خوشبندی) از تمام داده ها استفاده کنیم و سپس از نتایج آن استفاده کنیم تا داده هایی را برای برچسب زدن انتخاب کنیم که شامل اطلاعات مفید تری باشند.

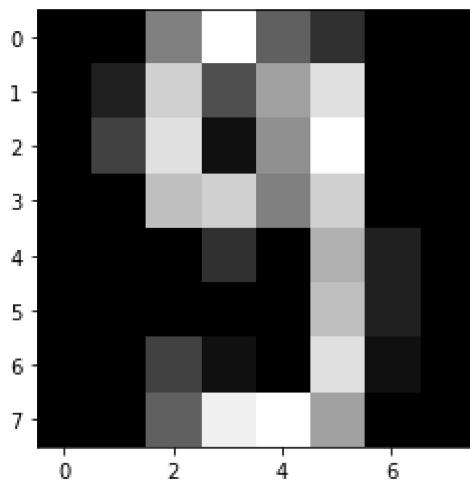
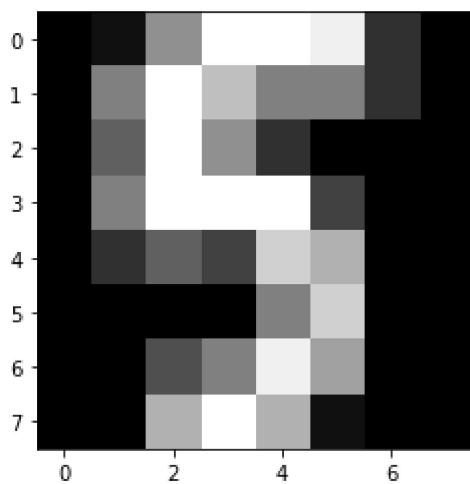
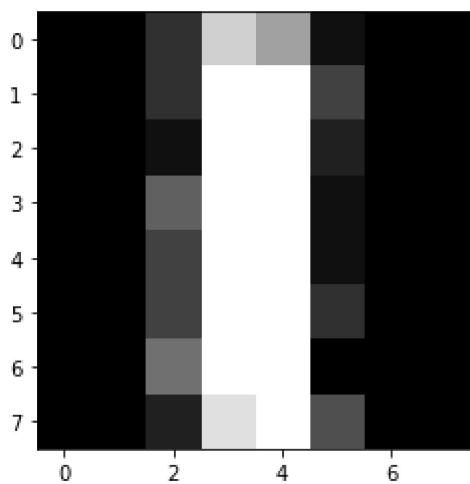
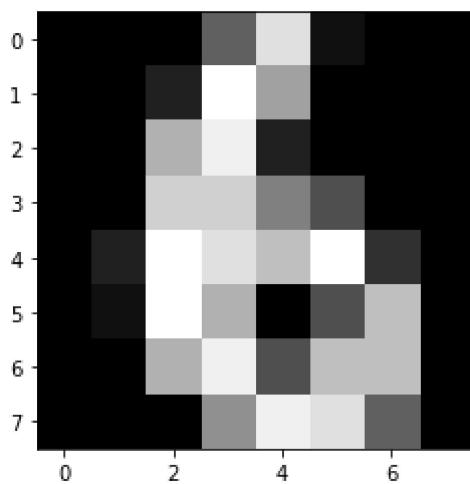
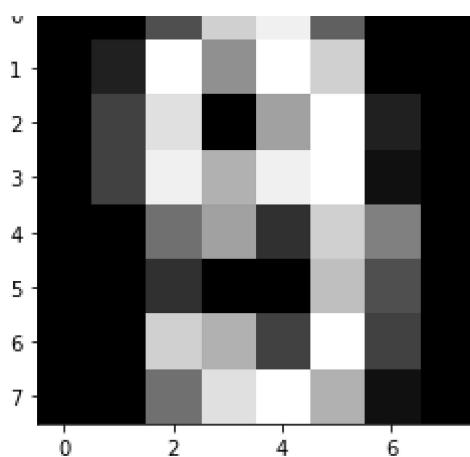
یک روش این است که داده ها را خوشبندی کنیم و سپس نزدیک ترین نمونه به مرکز هر خوشه را به عنوان نماینده هی آن خوشه انتخاب کنیم. سپس نماینده های هر خوشه را به طور دستی برچسب گذاری کنیم و از آن ها به عنوان داده برای یادگیری با ناظر خود استفاده کنیم.

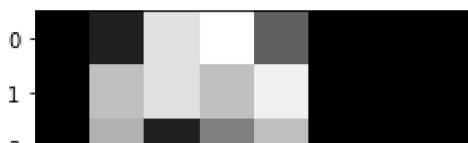
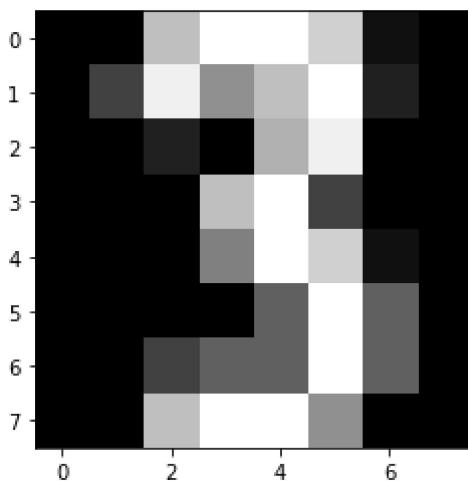
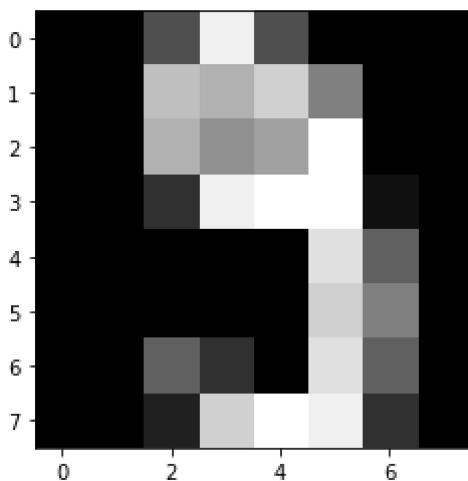
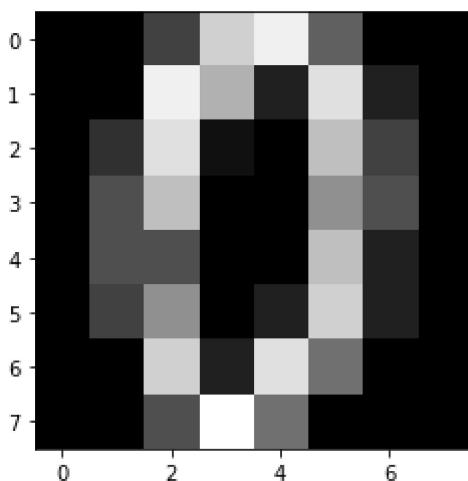
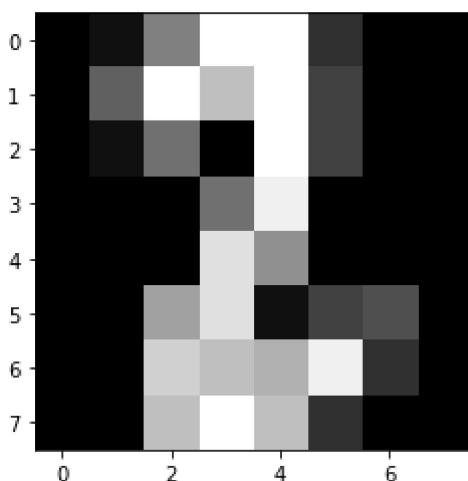
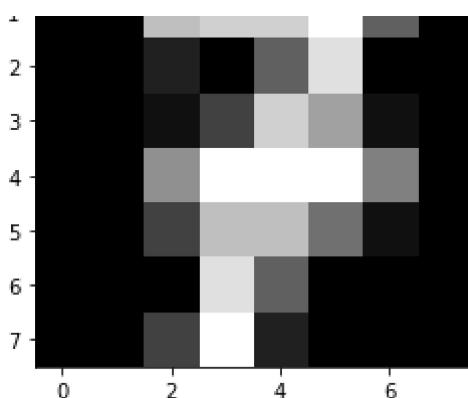
در مثال زیر با $k=50$ خوشبندی را انجام داده و سپس نماینده هی هر خوشه را نمایش داده ایم.

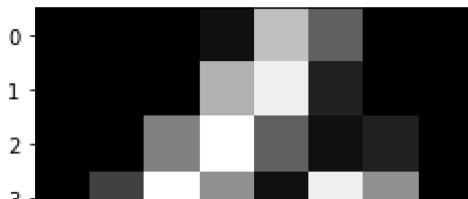
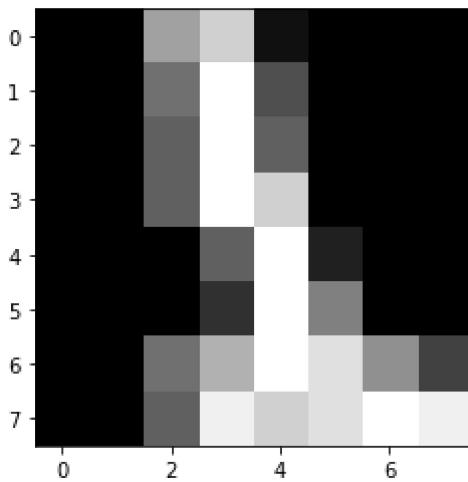
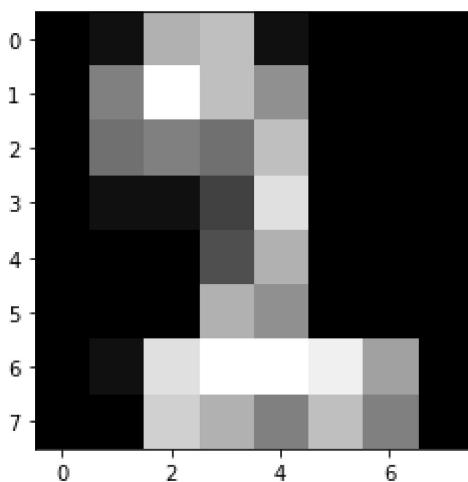
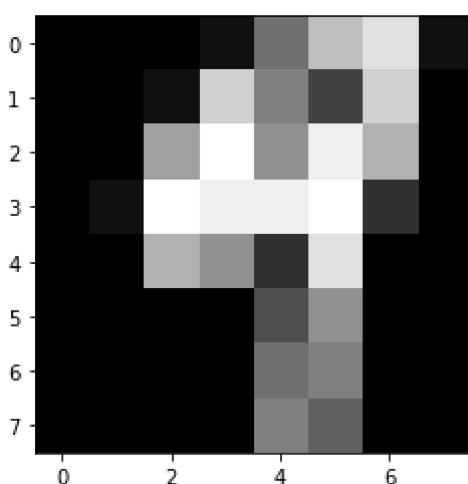
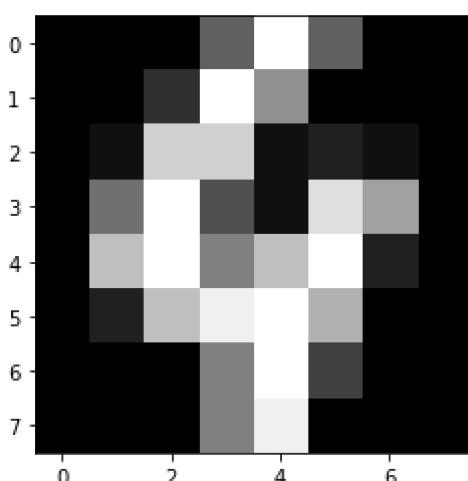
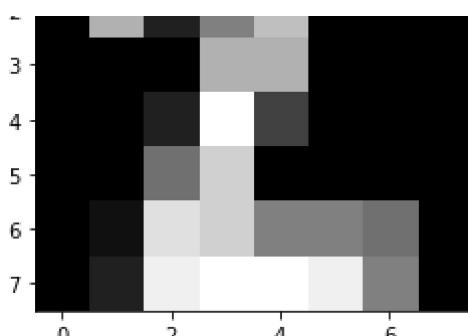
```
k = 50
kmeans = KMeans(n_clusters=k)
X_digits_dist = kmeans.fit_transform(X_train)
representative_digit_idx = np.argmin(X_digits_dist, axis=0)
XRepresentative_digits = X_train[representative_digit_idx]
YRepresentative_digits = y_train[representative_digit_idx]

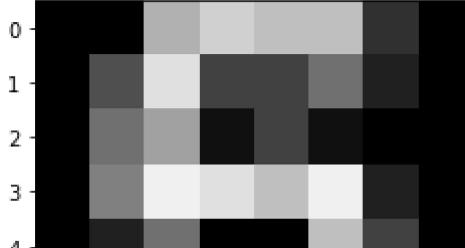
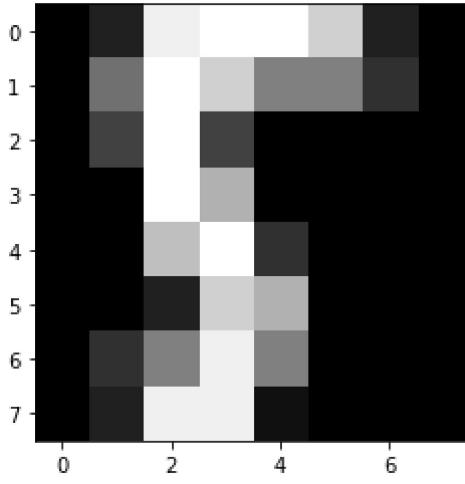
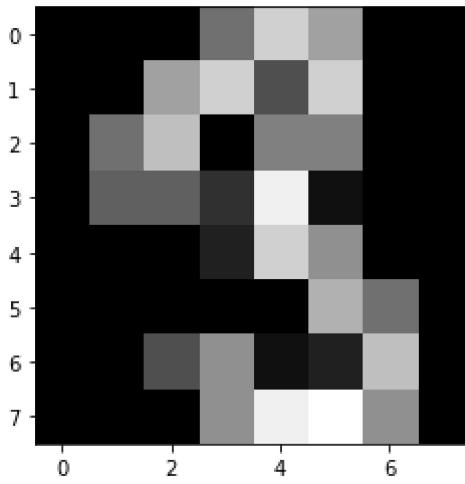
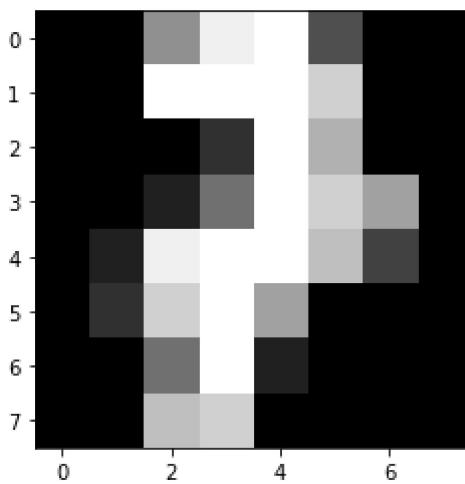
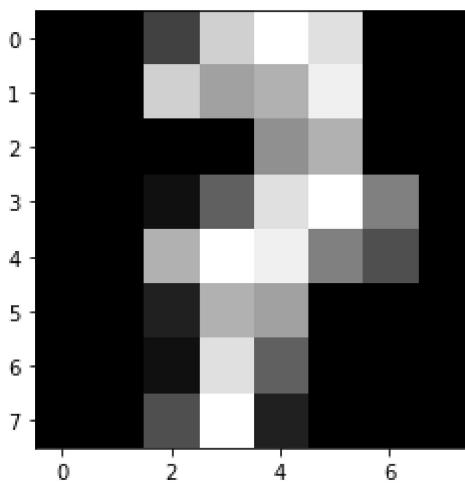
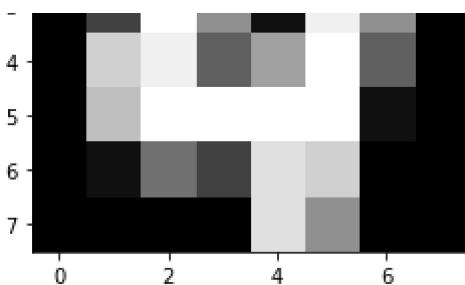
for i in range(50):
    X=XRepresentative_digits[i].reshape(8,8)
    plt.imshow(X,cmap='gray')
    plt.show()
```

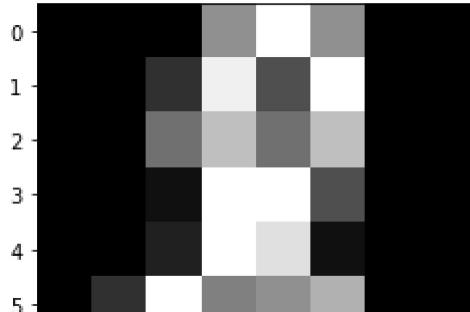
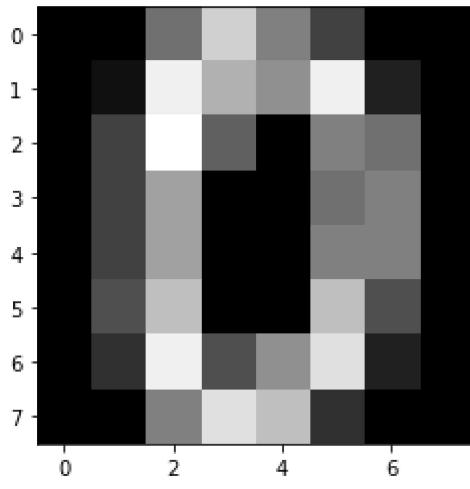
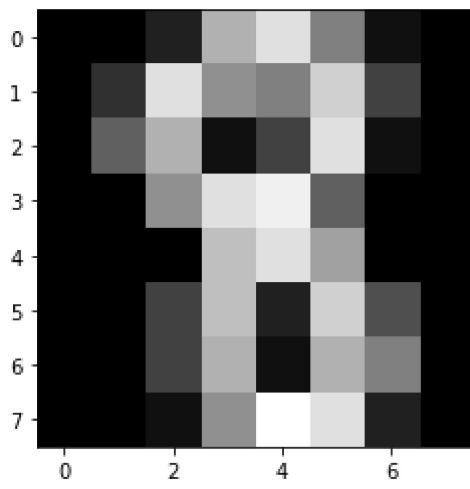
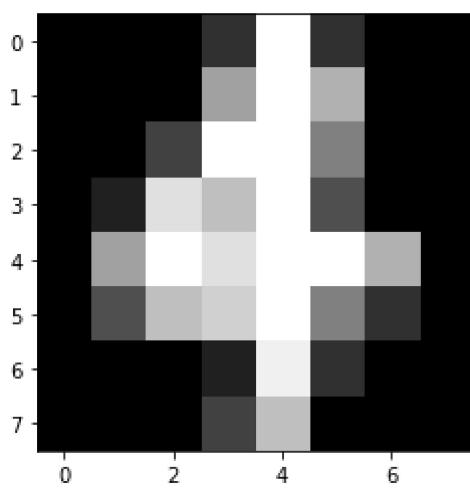
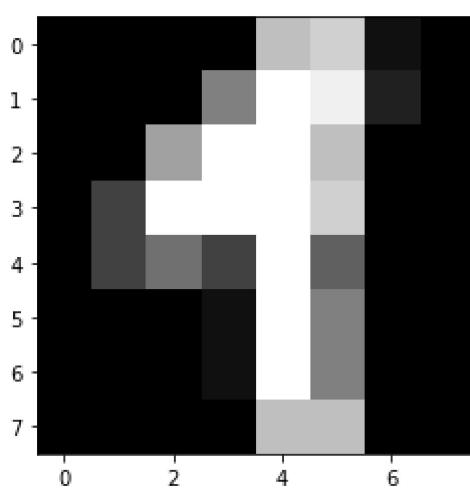
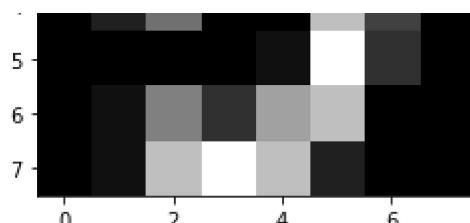


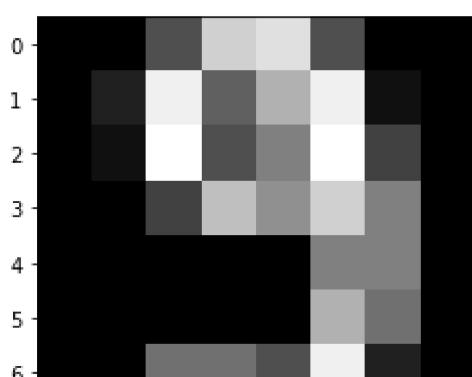
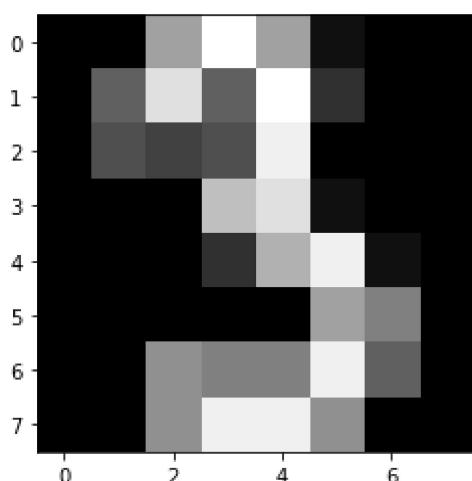
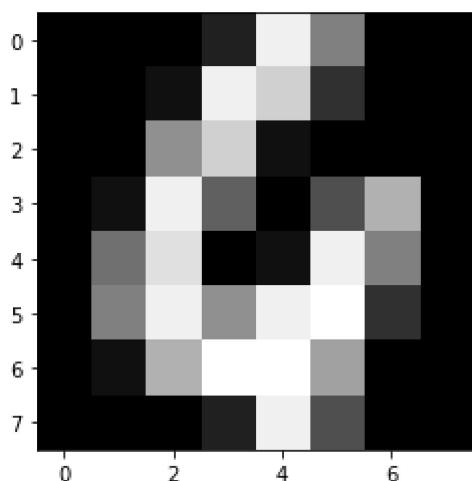
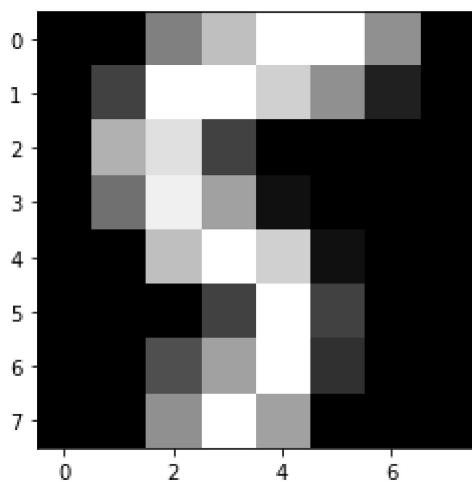
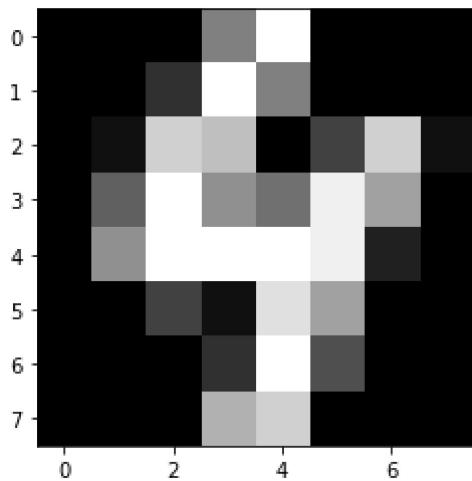
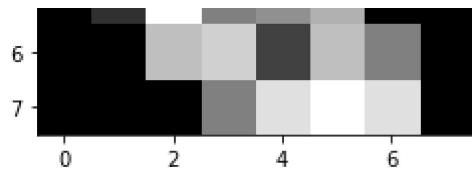


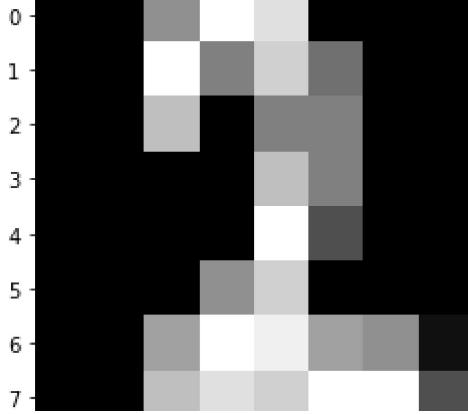
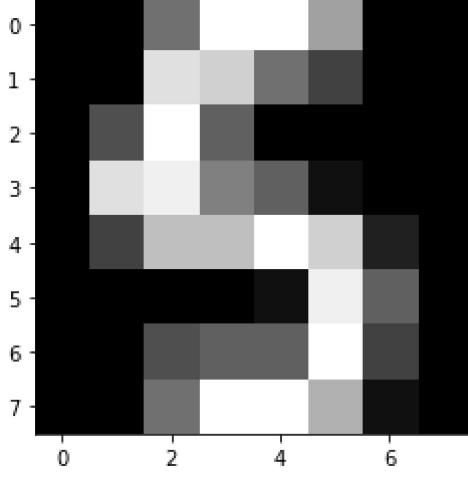
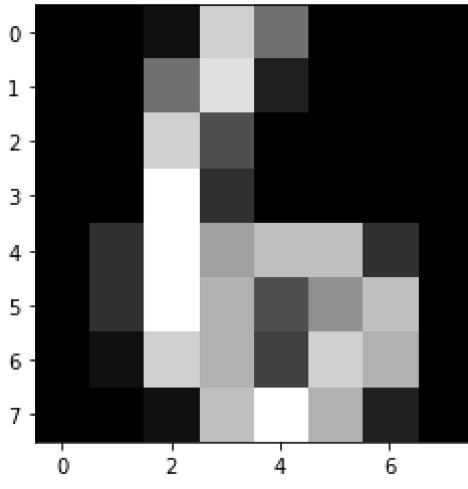
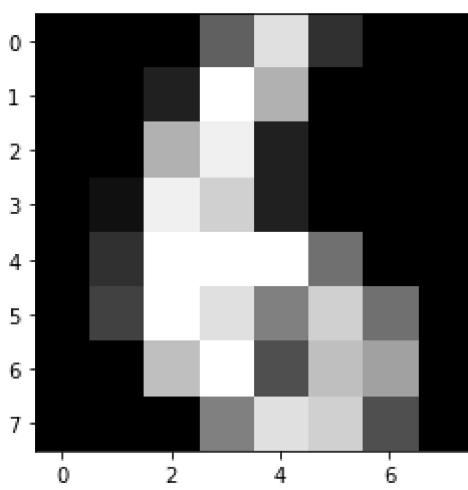
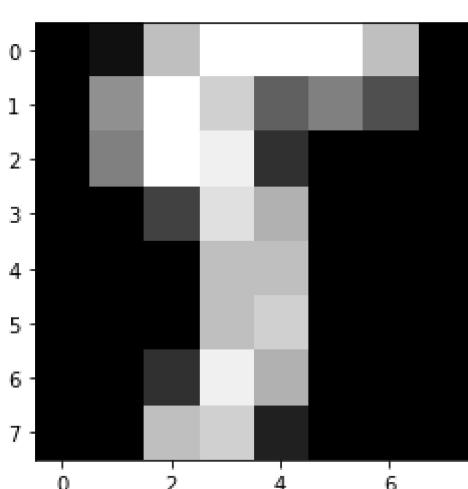
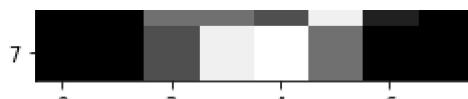


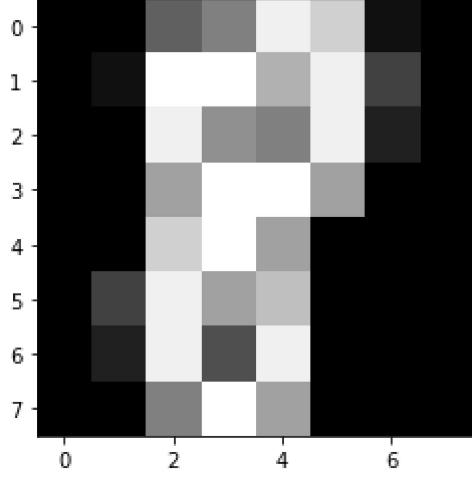
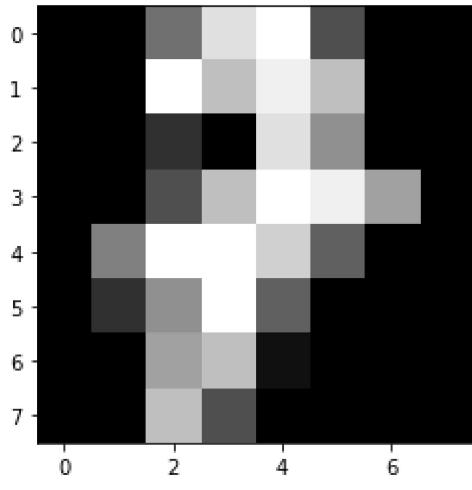
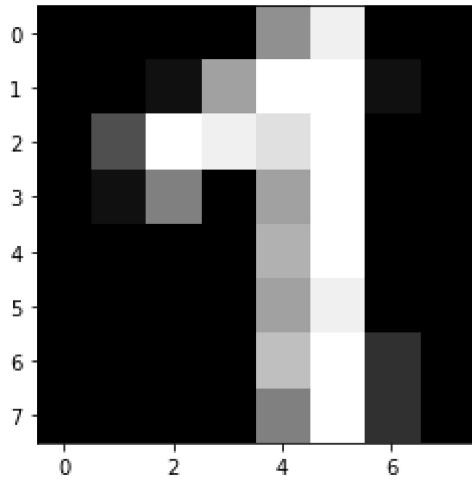
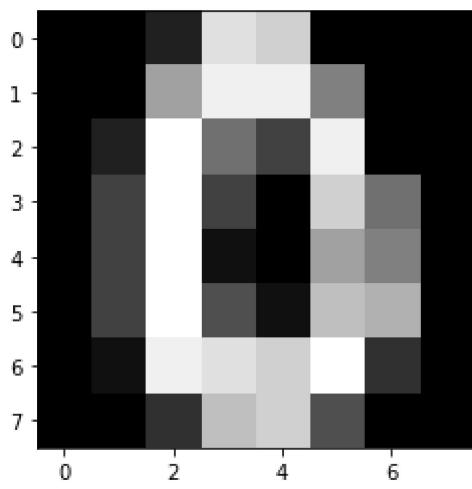
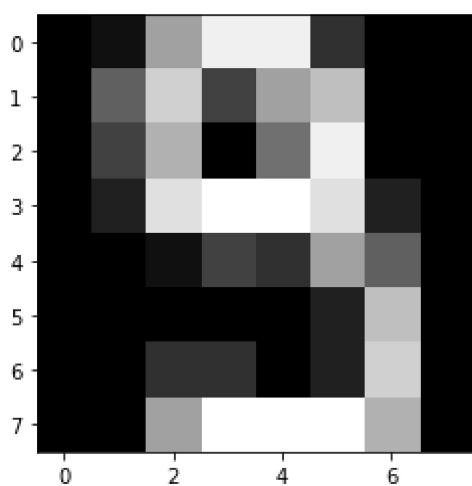












```
YRepresentativeDigits
```

```
array([0, 1, 3, 4, 7, 9, 6, 1, 5, 9, 7, 2, 0, 9, 3, 2, 4, 9, 2, 1, 4, 7,  
    7, 3, 5, 5, 1, 4, 8, 0, 8, 4, 5, 4, 3, 9, 5, 6, 6, 5, 2, 9, 0, 1,  
    7, 8, 4, 4, 9, 8])
```

```
4 [REDACTED] [REDACTED]
```

```
logReg = LogisticRegression()  
logReg.fit(XRepresentativeDigits, YRepresentativeDigits)  
logReg.score(XTest, yTest)
```

```
0.8844444444444445
```

میتوان دید استفاده از خوش بندی برای انتخاب داده هایی که برچسب میزنیم، باعث افزایش دقت شد.

اگر کمی این ایده را بسط دهیم به روش label propagation یا انتشار برچسب میرسیم. در این ایده بعد از برچسب زدن نماینده هر خوش، به کل اعضای آن خوش نیز برچسب نماینده اش را نسبت می دهیم و درواقع برچسبش را به کل خوش مننشر می کنیم و از کل داده ها برای آموزش به روش با ناظر خود استفاده خواهیم کرد.

```
6 [REDACTED] [REDACTED]
```

```
yTrainPropagated = np.empty(len(XTrain), dtype=np.int32)  
for i in range(k):  
    yTrainPropagated[kmeans.labels_==i] = YRepresentativeDigits[i]  
  
logReg = LogisticRegression()  
logReg.fit(XTrain, yTrainPropagated)  
logReg.score(XTest, yTest)
```

```
0.9133333333333333
```

```
4 [REDACTED] [REDACTED]
```

میتوان دید استفاده از label propagation، باعث افزایش دقت شد.

اشکال روش قبل در این است که توجهی به فاصله های نمونه ها با مرکز خوش و در نتیجه نماینده خوش نزدیک هستند به احتمال بالایی همان برچسب نماینده خود را خواهند داشت؛ اما در مورد نمونه های نزدیک به مرز خوشها نمی توان با همین اطمینان اظهار نظر کرد. به همین منظور در یک بهبود برای روش قبل، برچسب نماینده را تنها به درصد نزدیکی از نمونه ها به مرکز انتشار می دهیم. مثلاً فقط به 50 درصد نزدیک ترین نمونه ها به مرکز هر خوش برچسب نماینده آن خوش را نسبت می دهیم و سپس از تمام داده های برچسب گذاری شده برای آموزش استفاده می کنیم.

```
- [REDACTED] [REDACTED]
```

```
percentileClosest = 50  
XClusterDist = XDigitsDist[np.arange(len(XTrain)), kmeans.labels_]  
for i in range(k):  
    inCluster = (kmeans.labels_ == i)  
    clusterDist = XClusterDist[inCluster]  
    cutoffDistance = np.percentile(clusterDist, percentileClosest)  
    aboveCutoff = (XClusterDist > cutoffDistance)  
    XClusterDist[inCluster & aboveCutoff] = -1  
partiallyPropagated = (XClusterDist != -1)  
XTrainPartiallyPropagated = XTrain[partiallyPropagated]  
yTrainPartiallyPropagated = yTrainPropagated[partiallyPropagated]  
logReg = LogisticRegression()  
logReg.fit(XTrainPartiallyPropagated, yTrainPartiallyPropagated)  
logReg.score(XTest, yTest)
```

```
0.9155555555555556
```

میتوان دید انتشار برچسب تنها به نمونه های نزدیک تر به مرکز خوش که اطمینان بالایی دارند، اندکی دقت را بهبود داد.

4-3. یادگیری فعال (Active learning)

در این رویکرد ناظر با الگوریتم یادگیری ارتباط متفاصل دارد به این شکل که هر زمان الگوریتم برای داده هایی که اعلام می کند نیاز به برچسب داشت، ناظر آن داده ها را برایش برچسب گذاری کند.

یه روش یادگیری فعال uncertainty sampling نام دارد به این صورت که هر بارا لگوریتم با داده های برچسب داری که در اختیار دارد آموزش را انجام می دهد و روی داده های بدون برچسب پیش بینی انجام می دهد. با ازای هر پیش بینی یه ضریب اطمینان نیز محاسبه می کند و داده هایی را که به پیش بینی شان کمترین اطمینان را دارد به ناظر می دهد تا برایش برچسب بزند و این مراحل را تکرار می کند تا زمانی که بهبود قابل توجهی حاصل نشود.

روش های دیگر یادگیری فعال می توانند شامل موارد زیر باشد:

- الگوریتم داده هایی که موجب بیشترین کاهش در خطای باشد را برای برچسب زدن به ناظر بدهد
- چند الگوریتم روی داده ها اجرا شود و نمونه هایی که الگوریتم های مختلف پیش بینی های مختلفی برایشان داشته است برای برچسب زدن به ناظر داده شوند.

منابع

- Gel'ron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems (2nd ed.). O'Reilly.
- https://github.com/asharifiz/Introduction_to_Machine_Learning/blob/main/Slides/Chapter_02_Classical_Models/Clustering/section%202-3.pdf

