Agrupación: DBSCAN y K-Medoids

Minería de Datos

Zavala Roman Irvin Eduardo 1270771

K-Medoids

Es un derivado de KMeans, se basa en medoides que se definen como objetos representativos de un conjunto de datos donde la diferencia con los datos es mínima. Se caracteriza por ser más robusto al ruido y outliers que el KMeans.

class **sklearn_extra.cluster.KMedoids**(*n_clusters=8, metric='euclidean', method='alternate', init='heuristic', max_iter=300, random_state=None*) [source]

n_clusters : int, optional, default: 8
metric : string, or callable, optional, default: 'euclidean'
method : {'alternate', 'pam'}, default: 'alternate'
init : {'random', 'heuristic', 'k-medoids++', 'build'}, optional, default: 'build'
max_iter : int, optional, default : 300
random_state : int, RandomState instance or None, optional

Descripción del algoritmo

Alternado

- Seleccionar número de clusters
- Asignar cada elemento del dataset al medoide más cercano
- Dentro de los cluster se identifican nuevos medoides
- Repetir mientras los medoides cambien o hasta que se alcance max_iter

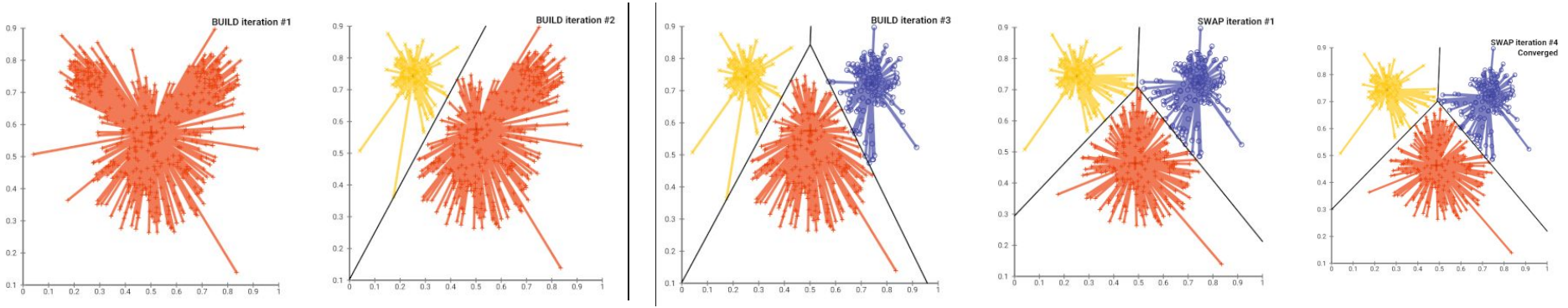
PAM (Partitioning Around Medoids)

- Seleccionar puntos donde la suma de las distancias a este de otros puntos sea la menor
- Repetir paso anterior hasta obtener n_clusters
- Asignar datos al medoide más cercano
- Calcular qué pasa si se cambian los mediodes por datos que no lo son, si los valores son mejores cambiar de medoide
- Repetir el paso anterior hasta que no haya cambios o se alcance max_iter

$$c = \sum_{Ci} \sum_{Pi \in Ci} |Pi - Ci|$$

Representacion de selección de medoides PAM

Build se refiere a la inicialización de los medoides y SWAP se refiere a la busca de mejores medoides.

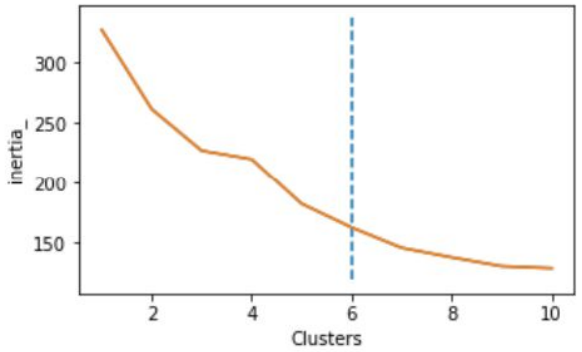


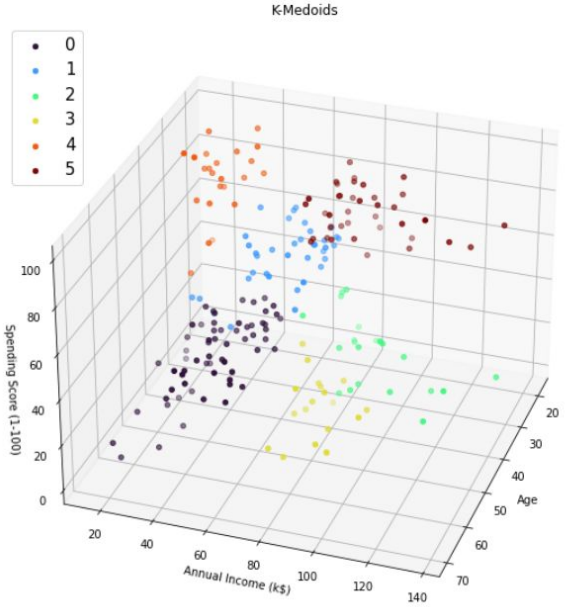
Ejemplo 1 - Dataset agrupación

```
dataset = pd.read_csv("customers_agr.csv", on_bad_lines='skip')
x = dataset.loc[:, ['Age',
                    'Annual Income (k$)',
                    'Spending Score (1-100)'
                    ]].values
scaler = StandardScaler().fit(x)
x_scaled = scaler.transform(x)
#elbow method
elbow = []
for i in range(1,11):
    kmedoids = KMedoids(n_clusters=i).fit(x_scaled)
    elbow.append(kmedoids.inertia_)
#plot elbow curve
plt.plot(np.arange(1,11),elbow)
plt.xlabel('Clusters')
plt.ylabel(inertia_)
plt.show()

kmedoids_opt = KMedoids(n_clusters=5).fit(x_scaled)
y = kmedoids opt.fit predict(x_scaled)
dataset['cluster'] = y

graficar_clusters(10, dataset)
```





Resumen evaluaciones

Extrínsecos

homogeneity_score: Que tantos datos de un cluster están en una clase

completeness_score: Que tantos datos de una clase están en el mismo cluster

v_measure_score: Promedio armónico entre los 2 anteriores

Intrínsecos

calinski_harabasz_score: Radio de la suma de la dispersión entre el cluster y dentro del cluster

silhouette_score: Relaciona la distancias dentro del cluster y la distancia entre clusters

davies_bouldin_score: Evalúa similaridad entre cada cluster con el más cercano, menos es mejor

Ejemplo 1 - Evaluación grupos (Intrínsecos)

```
 #--------------EVALUACION INSTRINSECA--------------
#Mayor es mejor
calinski = calinski_harabasz_score(x, y)
print("calinski_harabasz_score:",calinski)

#De -1 a 1, donde 1 es mejor
silhouette  = silhouette_score(x, y)
print("silhouette_score:",silhouette)

#Menor es mejor
davies  = davies_bouldin_score(x, y)
print("davies_bouldin_score:",davies)
```

n_clusters=6

```
calinski_harabasz_score: 105.49282013314067
silhouette_score: 0.3336335244519738
davies_bouldin_score: 1.2955718895697725
```

n_clusters=2

```
calinski_harabasz_score: 71.97231767882107
silhouette_score: 0.2617714484946135
davies_bouldin_score: 1.5102716888579146
```

n_clusters=3

```
calinski_harabasz_score: 91.01480895688285
silhouette_score: 0.314362951448644
davies_bouldin_score: 1.133734424788158
```

n_clusters=4

```
calinski_harabasz_score: 62.06152707912174
silhouette_score: 0.24891141946332937
davies_bouldin_score: 1.5004566838051132
```

n_clusters=5

```
calinski_harabasz_score: 93.46975286926086
silhouette_score: 0.3216891024612686
davies_bouldin_score: 1.4725309074051822
```

n_clusters=7

```
calinski_harabasz_score: 145.84516557449282
silhouette_score: 0.40456714699477453
davies_bouldin_score: 1.022579064765511
```

Ejemplo 2 - Dataset clasificación

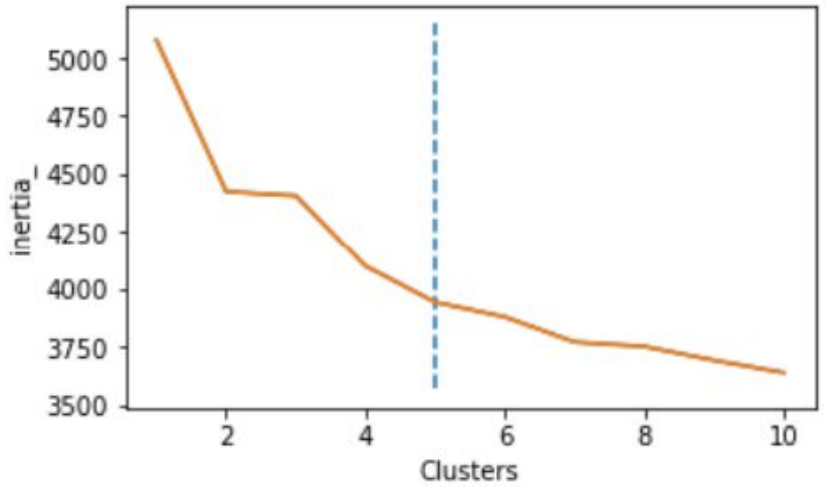
Clases

	A	B	C	D	E	F	G	H	I	J	K	L
1	fixed acidity	volatile acid	citric acid	residual sug	chlorides	free sulfur d	total sulfur d	density	pH	sulphates	alcohol	quality
2	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	bad
3	7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	bad
4	7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	bad
5	11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	good
6	7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	bad

```
dataset = pd.read csv("wine new.csv", on_bad_lines='skip')
labels = dataset.columns[0:11]
x = dataset.loc[:,labels].values
true_labels = dataset[['quality']].apply(lambda x: pd.factorize(x)[0])

scaler = StandardScaler().fit(x)
x scaled = scaler.transform(x)
#elbow method
elbow = []
for i in range(1,11):
    kmedoids = KMedoids(n clusters=i).fit(x_scaled)
    elbow.append(kmedoids.inertia_)
kn = KneeLocator(
    list(np.arange(1,11)),
    elbow,
    curve='convex',
    direction='decreasing',
    interp_method='polynomial',
)
plt.figure()
plt.plot(list(np.arange(1,11)), elbow)
plt.vlines(kn.knee, plt.ylim()[0], plt.ylim()[1], linestyles='dashed')
plt.plot(np.arange(1,11),elbow)
plt.xlabel('Clusters')
plt.ylabel('inertia_')
plt.show()

kmedoids opt = KMedoids(method = 'pam',n_clusters=5).fit(x_scaled)
y = kmedoids_opt.fit_predict(x_scaled)
```



Ejemplo 1 - Evaluación grupos (Extrínsecos)

```
#--------------EVALUACION EXTRINSECA--------------
 #0 a 1, mas alto mejor
homogeneo = metrics.homogeneity_score(true_labels, y)
print("homogeneity_score:",homogeneo)

#0 a 1, mas alto mejor
completeness = metrics.completeness_score(true_labels, y)
print("completeness_score:",completeness)

#0 a 1, mas alto mejor
vmeasure = metrics.v_measure_score(true_labels, y)
print("v_measure_score:",vmeasure)
```

n_clusters=2

```
homogeneity_score: 0.029166604117270286
completeness_score: 0.030521737145528065
v_measure_score: 0.02982878751398107
```

n_clusters=3

```
homogeneity_score: 0.06798594537423473
completeness_score: 0.044445027861616825
v_measure_score: 0.05375097536543772
```

n_clusters=4

```
homogeneity_score: 0.13511783165078026
completeness_score: 0.0706550991583821
v_measure_score: 0.09278930669656744
```

n_clusters=5

```
homogeneity_score: 0.1370335599315124
completeness_score: 0.06097214379930036
v_measure_score: 0.08439383072351381
```

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Es uno de los métodos de densidad, por lo que una característica principal es la capacidad de agrupar sin importar la forma. Se puede definir densidad como la cantidad de datos en un área.

```
class sklearn.cluster.DBSCAN(eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto',
leaf_size=30, p=None, n_jobs=None) ¶
```

```
eps : float, default=0.5
min_samples : int, default=5
metric : str, or callable, default='euclidean'
metric_params : dict, default=None
algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'
leaf_size : int, default=30
p : float, default=None
n_jobs : int, default=None
```

Descripción del algoritmo

1. Para cada dato, encontrar los datos en la “vecindad” dentro de la distancia *eps* para definir “núcleos” con *min_samples* vecinos
2. Definir grupos con “núcleos” conectados
3. Para los datos que no son “núcleos” al cluster más cercano si lo puede alcanzar, sino es un outlier

Notas:

- *eps* es el parámetro más importante ya que define las “vecindades”
- *min_samples* debe ser el adecuado para tener buenos “nucleos”
- En ningún momento se define la cantidad de clusters
- No se usa el método del codo, se grafican la distancia entre puntos

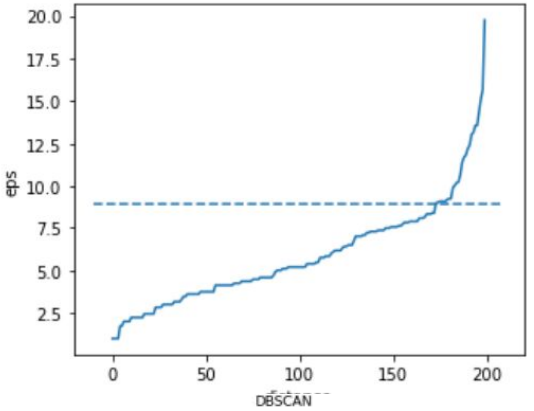
Ejemplo 1 - Dataset agrupación

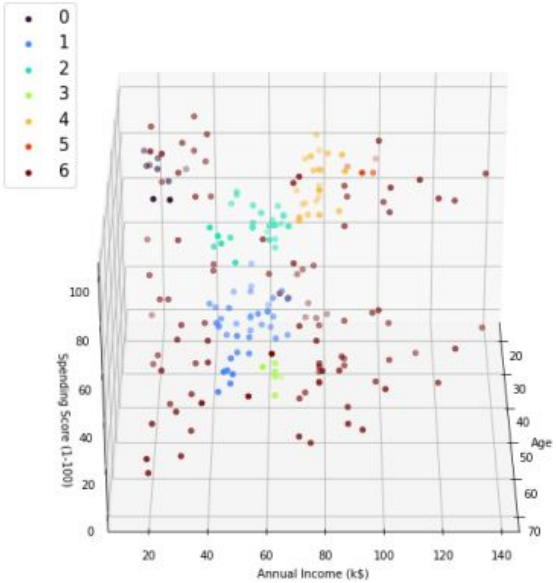
```
dataset = pd.read_csv("customers_agr.csv", on_bad_lines='skip')
labels = dataset.columns
x = dataset.loc[:, ['Age','Annual Income (k$)','Spending Score (1-100)']].values
neighb = NearestNeighbors(n_neighbors=2)
nbrs=neighb.fit(x)
distances,indices=nbrs.kneighbors(x)
distances = np.sort(distances, axis =0)
distances = distances[:,1]

kn = KneeLocator(
    np.arange(len(distances)),
    distances,
    curve='convex',
    direction='increasing',
    interp_method='polynomial',
)

plt.figure()
print(plt.ylim()[1])
plt.plot(np.arange(len(distances)), distances)
plt.hlines(distances[kn.knee], plt.xlim()[0], plt.xlim()[1], linestyles='dashed')
plt.xlabel("distance")
plt.ylabel("eps")
plt.show()

dbscan = DBSCAN(eps =9, min_samples = 6).fit(x)
y = dbscan.fit_predict(x)
dataset['cluster'] = y
graficar_clusters(1, dataset)
```





Ejemplo 1 - Evaluación grupos (Intrínsecos)

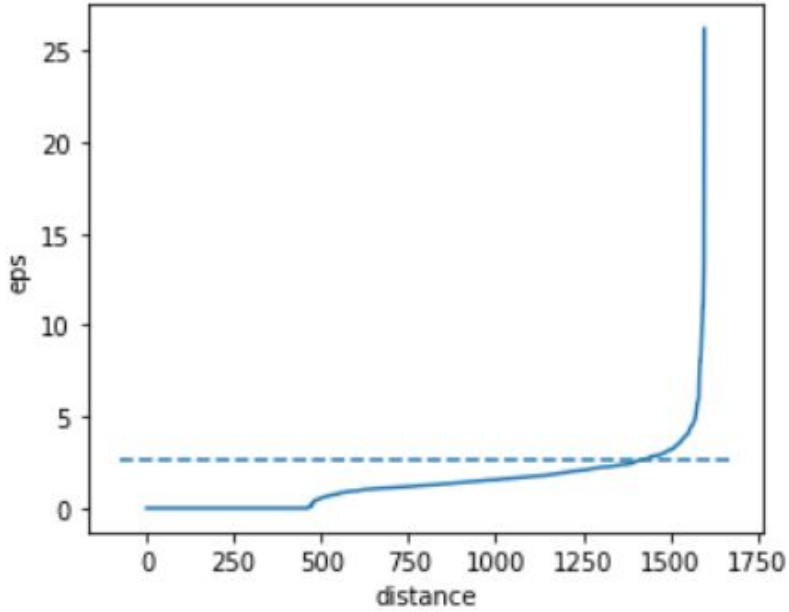
eps=8

```
_____________INSTRINSECO_____________
calinski_harabasz_score: 14.584352520329494
silhouette_score: 0.013736755228660 92
davies_bouldin_score: 1.715981081149743
```

eps=20

```
_____________INSTRINSECO_____________
calinski_harabasz_score: 7.7703804490152555
silhouette_score: 0.3588158894405232
davies_bouldin_score: 0.5864956418906484
```

Ejemplo 2 - Dataset clasificación



Ejemplo 1 - Evaluación grupos (Extrínsecos)

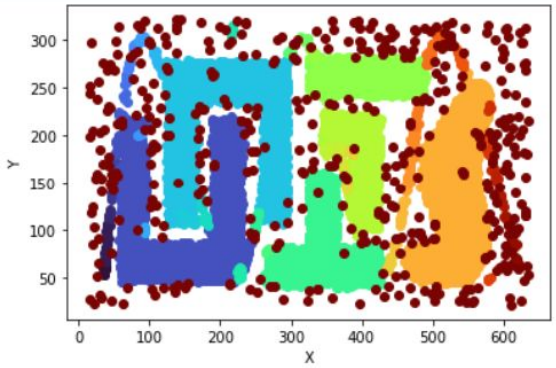
eps=2

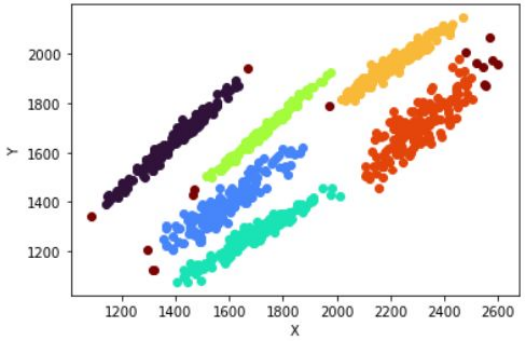
```
homogeneity_score: 0.018725065113929757
completeness_score: 0.010999053873914462
v_measure_score: 0.013857971707413557
```

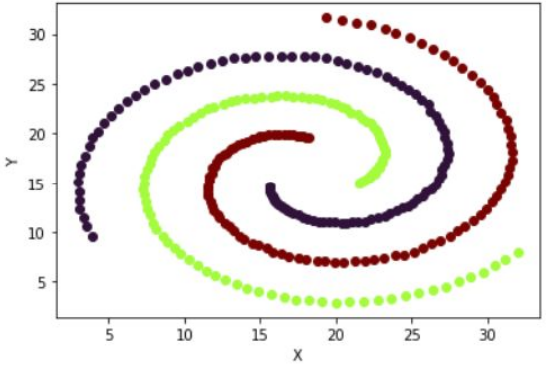
eps=25

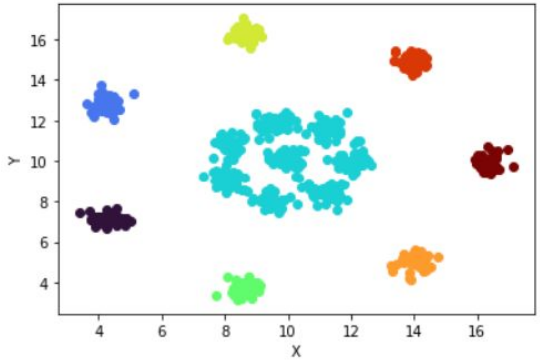
```
homogeneity_score: 0.0017026464333215587
completeness_score: 0.08613437089428044
v_measure_score: 0.0033392841389995012
```

Prueba DBSCAN en dataset de 2 dimensiones









Bibliografia

2. Clustering with KMedoids and Common-nearest-neighbors — scikit-learn-extra 0.2.0 documentation. (s/f). https://scikit-learn-extra.readthedocs.io/en/stable/modules/cluster.html

2.3. Clustering. (s/f). Scikit-Learn. https://scikit-learn.org/stable/modules/clustering.html

API reference. (s/f). Scikit-Learn. https://scikit-learn.org/stable/modules/classes.html

Awan, A. A. (s/f). *Implementing DBSCAN in python*. KDnuggets. https://www.kdnuggets.com/2022/08/implementing-dbscan-python.html

Mullin, T. (2020). *DBSCAN parameter estimation using python*. Medium. https://medium.com/@tarammullin/dbscan-parameter-estimation-ff8330e3a3bd

Scikit Learn - Clustering Performance Evaluation. (s/f). Tutorialspoint.com. https://www.tutorialspoint.com/scikit_learn/scikit_learn_clustering_performance_evaluation.htm