Clase15 IMA539

Mg. Alejandro Ferreira Vergara

May 19, 2023

1 K-means

- **Algoritmo de Aprendizaje No Supervisado**
- **Algoritmo de Clúster basados en Prototipos**

El **clustering basado en prototipos** significa que cada clúster está representado por un prototipo (un punto), que puede ser, el **centroide** (media) de puntos similares con características continuas o el **medioide** (el punto más representativo o más frecuente) en el caso de características categóricas.

K-means es muy bueno en la identificación de clusters con forma esférica, pero uno de los inconvenientes de este algoritmo de clustering es que tenemos que especificar el número de clusters, k, a priori. Una elección inadecuada de k puede dar lugar a un mal rendimiento de la agrupación.

[ ]:
```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

X, y = make_blobs(n_samples=150, n_features=2, centers=3, cluster_std=0.5,␣
 ↪shuffle=True, random_state=0)

plt.scatter(X[:, 0], X[:, 1], c='white', marker='o', edgecolor='black', s=50)
plt.grid()
plt.tight_layout()
#plt.savefig('images/11_01.png', dpi=300)
plt.show()
```

1.1 Algoritmo K-means

1. Elegir aleatoriamente k centroides (medioides en el caso de características discretas) $\mu^{(j)},\ j \in \{1, ..., k\}$ de los puntos de la muestra como centros de cluster iniciales.
2. Asignar cada muestra al centroide $\mu^{(j)}$ más cercano , $j \in \{1, ..., k\}$.
3. Mover los centroides $\mu^{(j)},\ j \in \{1, ..., k\}$ al centro de las muestras que le fueron asignadas.
4. Repetir los pasos 2 y 3 hasta que las asignaciones de clústeres no cambien o se alcance una tolerancia definida por el usuario o el número máximo de iteraciones.

Se requiere una medida de similutud: Podemos entender la similitud como una magnitud que es inversa de la distancia. Un magnitud frecuentemente utilizada para medir el inverso de la similitud es la siguiente distancia euclideana al cuadrado.

$$d(x,y)^2 = \sum_{j=1}^{m} (x_j - y_j)^2 = \|x - y\|_2^2$$

Sobre la base de esta métrica de distancia, podemos describir el algoritmo de K-means como un problema de minimización; un enfoque iterativo para minimizar la **Suma de Errores Cuadrados (SSE)** dentro del clúster, que a veces también se denomina **inercia del clúster**:

$$SSE = \sum_{i=1}^{n} \sum_{j=1}^{k} w^{(i,j)} \|x^{(i)} - \mu^{(j)}\|_2^2$$

$\mu^{(j)}$: Centroide del clúster j

$w^{(i,j)} = 1$ si la muestra $x^{(i)}$ está en el cluster j

$w^{(i,j)} = 0$ en caso contrario.

[ ]:
```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=3, init='random', n_init=10, max_iter=300, tol=1e-04,␣
 ↪random_state=0)

y_km = km.fit_predict(X)
```

¿A qué elementos hacen referencia cada uno de los atributos utilizados en el algoritmos *Kmeans()* de sk-learn?

NOTA: Cuando aplicamos K-means a datos del mundo real utilizando una métrica de distancia euclidiana, debemos asegurarnos de que las características se miden en la misma escala.

[ ]:
```
plt.scatter(X[y_km == 0, 0],
            X[y_km == 0, 1],
            s=50, c='lightgreen',
            marker='s', edgecolor='black',
            label='cluster 1')
plt.scatter(X[y_km == 1, 0],
            X[y_km == 1, 1],
            s=50, c='orange',
            marker='o', edgecolor='black',
            label='cluster 2')
plt.scatter(X[y_km == 2, 0],
            X[y_km == 2, 1],
            s=50, c='lightblue',
            marker='v', edgecolor='black',
            label='cluster 3')
plt.scatter(km.cluster_centers_[:, 0],
            km.cluster_centers_[:, 1],
            s=250, marker='*',
```

```
            c='red', edgecolor='black',
            label='centroides')
plt.legend(scatterpoints=1)
plt.grid()
plt.tight_layout()
#plt.savefig('images/11_02.png', dpi=300)
plt.show()
```

¿Qué sucede si aumentamos el número de clusters k?

[ ]:

1.2 Una versión mejorada: K-means ++

1. Inicializar un conjunto vacío **M** para almacenar los k centroides que se seleccionan.
2. Elegir aleatoriamente el primer centroide $\mu^{(j)}$ de las muestras de entrada y asignarlo a **M**.
3. Para cada muestra $x^{(i)}$ que no esté en **M**, encontrar la mínima distancia al cuadrado $d\left(x^{(i)}, M\right)^2$ a cualquiera de los centroides en **M**.
4. Para seleccionar aleatoriamente el siguiente centroide $\mu^{(p)}$, utilice una distribución de probabilidad ponderada igual a $\dfrac{d\left(\mu^{(p)}, M\right)^2}{\sum_i d\left(x^{(i)}, M\right)^2}$
5. Repetir los pasos 2 y 3 hasta elegir k centroides.
6. Proceder con el algoritmo clásico de k-means.

[ ]:
```
km = KMeans(n_clusters=3, init='k-means++', n_init=10, max_iter=300, tol=1e-04,␣
 ↪random_state=0)

y_km = km.fit_predict(X)

plt.scatter(X[y_km == 0, 0],
            X[y_km == 0, 1],
            s=50, c='lightgreen',
            marker='s', edgecolor='black',
            label='cluster 1')
plt.scatter(X[y_km == 1, 0],
            X[y_km == 1, 1],
            s=50, c='orange',
            marker='o', edgecolor='black',
            label='cluster 2')
plt.scatter(X[y_km == 2, 0],
            X[y_km == 2, 1],
            s=50, c='lightblue',
            marker='v', edgecolor='black',
            label='cluster 3')
plt.scatter(km.cluster_centers_[:, 0],
```

```
            km.cluster_centers_[:, 1],
            s=250, marker='*',
            c='red', edgecolor='black',
            label='centroides')
plt.legend(scatterpoints=1)
plt.grid()
plt.tight_layout()
#plt.savefig('images/11_02.png', dpi=300)
plt.show()
```

2 Métricas para evaluar modelos de Clúster

2.1 Método del Codo

Uno de los principales retos del aprendizaje no supervisado es que no conocemos la respuesta definitiva. No tenemos las etiquetas de clase verdaderas en nuestro conjunto de datos que nos permitan evaluar el rendimiento de un modelo.

Por lo tanto, para cuantificar la calidad de la agrupación, necesitamos utilizar métricas intrínsecas, como la SSE (distorsión) dentro del clúster, para comparar el rendimiento de diferentes agrupaciones de K-means.

[ ]:
```
print('Distorción: %.2f' % km.inertia_)
```

Intuitivamente, podemos decir que, si k aumenta, la distorsión disminuirá. Esto se debe a que las muestras estarán más cerca de los centroides a los que están asignadas. La idea que subyace al método del codo es identificar el valor de k en el que la distorsión comienza disminuir más lentamente.

[ ]:
```
distortions = []
for i in range(1, 11):
    km = KMeans(n_clusters=i, init='k-means++', n_init=10, max_iter=300,␣
 ↪random_state=0)
    km.fit(X)
    distortions.append(km.inertia_)

plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Cantidad de clusters')
plt.ylabel('Distorción')
plt.tight_layout()
#plt.savefig('images/11_03.png', dpi=300)
plt.show()
```

2.2 Gráficos de Silueta

El análisis de la silueta puede utilizarse como una herramienta gráfica para trazar una medida de lo estrechamente agrupadas que están las muestras en los clusters (calidad de la agrupación).

Para calcular el **coeficiente de silueta** (s_i) de una sola muestra en nuestro conjunto de datos, podemos aplicar los siguientes tres pasos:

1. Calcular la **cohesión del clúster** $a^{(i)}$ como la distancia media entre una muestra $x^{(i)}$ y todos los demás puntos del mismo clúster.
2. Calcular la **separación entre clusters** $b^{(i)}$ y el siguiente cluster más cercano como la distancia media entre la muestra $x^{(i)}$ y todas las muestras del cluster más cercano.
3. Calcular calcular el coeficiente de silueta $s^{(i)}$ como la diferencia entre la cohesión y la separación del clúster dividida por la mayor de las dos:

$$s^{(i)} = \frac{b^{(i)} - a^{(i)}}{max\left\{b^{(i)}, a^{(i)}\right\}}$$

Finalmente, el coeficiente de silueta para un conjunto de muestras se calcula como la media del coeficiente de silueta de cada muestra.

[ ]:
```
import numpy as np
from matplotlib import cm
from sklearn.metrics import silhouette_samples

km = KMeans(n_clusters=3, init='k-means++',n_init=10,␣
 ↪max_iter=300,tol=1e-04,random_state=0)
y_km = km.fit_predict(X)

cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(X, y_km, metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height=1.0,
             edgecolor='none', color=color)

    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
```

```
#plt.savefig('images/11_04.png', dpi=300)
plt.show()
```

- El coeficiente de silueta está acotado en el intervalo de -1 a 1.
- El coeficiente de silueta es 0 si la separación y la cohesión de los clusters son iguales ($b^{(i)} = a^{(i)}$).
- Además, nos acercamos a un coeficiente de silueta ideal de 1 si $b^{(i)} >> a^{(i)}$, ya que $b^{(i)}$ cuantifica el grado de disimilitud de una muestra con respecto a otros clusters, y $a^{(i)}$ nos dice el grado de similitud con las demás muestras de su propio cluster.

[ ]:
```
km = KMeans(n_clusters=2,init='k-means++',n_init=10,max_iter=300,tol=1e-04,random_state=0)
y_km = km.fit_predict(X)

plt.scatter(X[y_km == 0, 0],
            X[y_km == 0, 1],
            s=50,
            c='lightgreen',
            edgecolor='black',
            marker='s',
            label='cluster 1')
plt.scatter(X[y_km == 1, 0],
            X[y_km == 1, 1],
            s=50,
            c='orange',
            edgecolor='black',
            marker='o',
            label='cluster 2')

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1],
            s=250, marker='*', c='red', label='centroids')
plt.legend()
plt.grid()
plt.tight_layout()
#plt.savefig('images/11_05.png', dpi=300)
plt.show()
```

[ ]:
```
cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(X, y_km, metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
```

```
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals,
             height=1.0,edgecolor='none', color=color)
    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_06.png', dpi=300)
plt.show()
```

[ ]:
```
km = KMeans(n_clusters=4,init='k-means++',n_init=10,max_iter=300,tol=1e-04,random_state=0)
y_km = km.fit_predict(X)

plt.scatter(X[y_km == 0, 0],
            X[y_km == 0, 1],
            s=50,
            c='lightgreen',
            edgecolor='black',
            marker='s',
            label='cluster 1')
plt.scatter(X[y_km == 1, 0],
            X[y_km == 1, 1],
            s=50,
            c='orange',
            edgecolor='black',
            marker='o',
            label='cluster 2')

plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1],
            s=250, marker='*', c='red', label='centroids')
plt.legend()
plt.grid()
plt.tight_layout()
#plt.savefig('images/11_05.png', dpi=300)
plt.show()
```

[ ]:
```
cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(X, y_km, metric='euclidean')
y_ax_lower, y_ax_upper = 0, 0
```

```
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals,
             height=1.0,edgecolor='none', color=color)
    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)

silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color="red", linestyle="--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()
#plt.savefig('images/11_06.png', dpi=300)
plt.show()
```