

Reporte practica 2

Quiñones Mayorga Rodrigo

Introducción

Es esta practica vamos a realizar el algoritmo de K-mans, y en la pagina dada por el profesor se nos describe el funcionamiento de el mismo, siendo esta una técnica para agrupar en grupos, que aquí se llaman clusters según que similitudes contengan los datos.

Síntesis

Organiza datos en un número predeterminado de clústeres (k). El algoritmo busca minimizar la suma de las distancias al cuadrado entre los puntos de datos y el centroide del clúster al que pertenecen.

Luego se itera ajustando los centroides y mandando a los datos a los grupos mas cercanos, hasta que no cambien de manera significativa los centroides.

```
In [1]: runfile('/Users/rodrigo/.spyder-py3/temp.py', wdir='/Users/rodrigo/.spyder-py3')
Ingresar el número de puntos de datos: 25
Ingresar el número de clústeres: 8
Puntos generados:
[[8.62771275 9.78457217]
 [9.49177607 3.20306283]
 [8.37180958 7.08254584]
 [4.89952429 4.24244328]
 [2.44747576 4.49831257]
 [0.52058314 4.62128704]
 [4.36693274 0.05622786]
 [0.08495375 2.63271516]
 [5.79577018 7.75992809]
 [4.99572806 9.12482726]
 [7.78014111 1.55385134]
 [1.53957556 6.01543474]
 [6.21636241 2.13112076]
 [1.20274437 7.34136883]
 [8.08836518 4.34683867]
 [1.56878743 0.71747455]
 [9.93006858 1.81234831]
 [2.09794969 1.3640902 ]
 [7.93776419 1.84380889]
 [9.23045545 7.51216469]
 [7.08388146 9.05534866]
 [5.87632925 8.6747682 ]
 [6.50299767 8.43657201]
 Score: -37.571183799113555
```

```
In [2]: runfile('/Users/rodrigo/.spyder-py3/temp.py', wdir='/Users/rodrigo/.spyder-py3')
Ingresar el número de puntos de datos: 27
Ingresar el número de clústeres: 27
Puntos generados:
[[8.00369537 7.19862922]
 [4.14233891 8.2494982 ]
 [0.41277626 6.68653519]
 [1.19725794 6.77479639]
 [8.57047925 7.8666617 ]
 [6.3058366 4.32614426]
 [2.06404026 9.52024373]
 [2.85840935 7.55878012]
 [5.38705159 2.93782453]
 [4.72002569 8.71142472]
 [3.76502099 6.29236904]
 [2.63130908 4.98858953]
 [2.55223854 4.33182207]
 [8.37323762 6.59499976]
 [1.14710028 9.73442829]
 [4.90560754 8.55503829]
 [6.11132916 7.76145053]
 [1.41477651 4.93997229]
 [0.1752129 3.1523151 ]
 [6.79120301 3.43930013]
 [0.10501854 6.8340036 ]
 [4.80476272 7.15753552]
 [1.63754676 4.27590976]
 [9.53144053 7.91750288]
 [0.5226804 7.6615873 ]
 [3.05259814 6.83145003]
 [9.93489025 1.86940579]
 Score: -1.4791141972893971e-31
```

El primer ejemplo que implemente fue el de SCORE, que básicamente mide que tan bien están agrupados los valores, aquí el score mas cercano a 0 significa que esta mejor agrupado, mientras que este menor sea al numero negativo, significa que esta mas disperso dentro de los grupos.

Lo que realice fue un código donde el usuario ingresa el numero de datos ademas de el numero de grupos, luego genera números aleatorios y te da finalmente el score.

El segundo código fue tomando el ejemplo de predicciones, donde primero se generan unos datos, se agrupan en cluster para posteriormente que el usuario ingrese nuevos números los cuales se agregaran a grupos usando la información anterior con la que se entrenó.

```
In [1]: runfile('/Users/rodrigo/.spyder-py3/temp.py', wdir='/Users/rodrigo/.spyder-py3')
¿Cuántos puntos de datos aleatorios deseas generar? 50
¿Cuántas características (dimensiones) tendrán los puntos? 2

Puntos de datos generados aleatoriamente:
[[7.49852445 1.87862828]
 [5.73755375 0.61742353]
 [7.47750158 8.27134479]
 [1.38484315 3.24136563]
 [6.21255098 5.58144661]
 [1.88028293 5.8642179 ]
 [5.97499158 0.60822525]
 [6.60511352 9.43515638]
 [2.83032622 3.33418089]
 [3.33288559 8.54425338]
 [9.90845501 9.55084154]
 [9.95856901 5.98013733]
 [0.1202335 2.64966616]
 [6.4375 2.47446539]
 [2.71291145 8.19061037]
 [2.50087505 2.33085535]
 [7.58415845 3.51120422]
 [1.4422344 9.93229378]
 [6.8726425 1.39569014]
 [7.99543831 7.77599434]
 [5.71717635 1.33526882]
 [9.94150684 5.44387418]
 [3.48325331 7.94738399]
 [7.21361253 8.83651746]
 [8.54406359 9.07502409]
 [5.26976956 7.67008514]
 [1.06813185 8.43575324]
 [7.9912515 7.98780837]
 [8.52870414 2.45956006]
 [1.70049931 9.25245088]
 [8.99280762 4.11380081]
 [7.37689974 8.5985175 ]
 [3.33718542 4.56474201]
 [0.14589084 8.91564481]
 [5.40076352 9.45681206]
 [7.93894894 0.1083387 ]
 [6.01797175 9.77179787]
 [6.42495662 6.39078788]
 [8.23649785 5.91922741]
 [4.71783903 0.89538774]
 [3.24808546 8.27070021]
 [7.3714643 1.18136668]
 [3.5415084 2.37336387]
 [0.93762505 7.00294467]
 [9.6562685 1.11861763]
 [8.52638773 1.66033981]
 [0.31347293 4.37729446]
 [6.69183457 5.59990827]
 [8.76897125 8.89684203]
 [6.44412615 9.80262645]]

¿Cuántos clústeres quieres formar? 8

Centroides del modelo:
[[8.29465443 8.59376752]
 [7.08150011 1.31110934]
 [1.00798267 7.55464015]
 [2.00406352 3.26735262]
 [6.72837515 5.27083675]
 [2.65331159 8.68961543]
 [9.28234533 5.36425993]
 [6.1585595 9.1621659 ]]

Introduce nuevas muestras para predecir su clúster:
¿Cuántas nuevas muestras deseas predecir? 5
Ingresa las coordenadas de la muestra 1 (separadas por espacio): 1 5
Ingresa las coordenadas de la muestra 2 (separadas por espacio): 5 4
Ingresa las coordenadas de la muestra 3 (separadas por espacio): 4 8
Ingresa las coordenadas de la muestra 4 (separadas por espacio): 4 2
Ingresa las coordenadas de la muestra 5 (separadas por espacio): 1 1

Predicciones de las nuevas muestras (índices de clúster):
[3 4 5 3 3]
```

Este código genera puntos de datos aleatorios, entrena un modelo KMeans con ellos, y permite al usuario predecir a qué clústeres pertenecen nuevas muestras.

Para nuestro tercer código, haremos el características polinomiales, el cual transforma características en combinaciones mas complejas, haremos el GET_FEATURE_NAMES

La razón por la cual hacemos esto es para darle más "información" al modelo que estamos entrenando.

Cómo funciona el algoritmo en el ejemplo

En el código que te di, los pasos son los siguientes:

1. **Generar datos aleatorios:** En lugar de decir manualmente qué características tienen nuestras "pelotas", el programa genera números aleatorios para simular las características (tamaño, peso, etc.).
2. **Transformar las características:** Después, aplicamos el algoritmo **PolynomialFeatures**, que toma las características originales y genera nuevas combinaciones de ellas (cuadrados, productos, etc.).
3. **Obtener los nombres de las características nuevas:** Luego, usamos la función `get_feature_names_out()` para obtener los nombres de estas nuevas características, por ejemplo: "Tamaño^2", "Tamaño x Peso", etc.
4. **Transformar los datos:** Finalmente, los datos originales (que eran simplemente el tamaño y peso de las pelotas) se transforman en un nuevo conjunto de datos que incluye todas las combinaciones nuevas de las características. Ahora tenemos más información con la que el modelo puede trabajar.

Repositorio

<https://github.com/RodQM/Practica2-Machine-LEARNING>

```
¿que grado de polinomio deseas usar? 2
```

```
Características originales:
```

```
[[6.35967055 4.90291411 9.28072081]
 [6.0910561 4.13182799 8.47821944]
 [3.74316339 4.17633193 3.97324385]
 [8.45381531 4.09434206 8.62695603]
 [5.78932007 6.17247248 7.80219543]
 [2.79151814 6.02416624 8.54094799]
 [2.40342476 2.29928106 7.09468667]
 [7.2878532 5.36384673 8.50343336]
 [9.03458042 7.44295378 8.29175996]
 [4.53190192 6.84108263 6.19117021]
 [3.24331257 6.06015807 6.81395173]
 [9.40271294 8.9310397 1.42997275]
 [8.55562509 7.57911831 3.27269481]
 [0.52056868 5.56736515 4.97327454]
 [9.03727336 2.75430972 8.808669 ]]
```

```
Nombres de las características transformadas:
```

```
['1' 'x0' 'x1' 'x2' 'x0^2' 'x0 x1' 'x0 x2' 'x1^2' 'x1 x2' 'x2^2']
```

```
Datos transformados:
```

```
[[ 1. 6.35967055 4.90291411 9.28072081 40.44540949 31.18091847
 59.02232682 24.03856677 45.50257702 86.13177879]
 [ 1. 6.0910561 4.13182799 8.47821944 37.10096442 25.16719606
 51.64131025 17.0720025 35.03054435 71.88020489]
 [ 1. 3.74316339 4.17633193 3.97324385 14.01127217 15.63269277
 14.87250091 17.44174835 16.59358512 15.78666666]
 [ 1. 8.45381531 4.09434206 8.62695603 71.46699328 34.6128116
 72.93069296 16.76363692 35.32170894 74.42437036]
 [ 1. 5.78932007 6.17247248 7.80219543 33.51622685 35.7344188
 45.16940656 38.09941652 48.15883656 60.87425348]
 [ 1. 2.79151814 6.02416624 8.54094799 7.79257352 16.81656933
 23.84221125 36.29057884 51.45209052 72.94779261]
 [ 1. 2.40342476 2.29928106 7.09468667 5.77645056 5.52614901
 17.05154559 5.28669338 16.31267867 50.33457898]
 [ 1. 7.2878532 5.36384673 8.50343336 53.1128043 39.09092759
 61.97177401 28.77085177 45.61111322 72.30837884]
 [ 1. 9.03458042 7.44295378 8.29175996 81.62364343 67.2439645
 74.91257224 55.39756095 61.71518615 68.75328329]
 [ 1. 4.53190192 6.84108263 6.19117021 20.53813498 31.00311547
 28.05777612 46.80041151 42.35430693 38.33058851]
 [ 1. 3.24331257 6.06015807 6.81395173 10.5190764 19.65498681
 22.09977529 36.72551578 41.29362456 46.42993824]
 [ 1. 9.40271294 8.9310397 1.42997275 88.41101058 83.97600248
 13.44562323 79.76347004 12.77114335 2.04482205]
 [ 1. 8.55562509 7.57911831 3.27269481 73.19872071 64.84409483
 27.99994984 57.44303443 24.80414118 10.71053133]
 [ 1. 0.52056868 5.56736515 4.97327454 0.27099175 2.89819595
 2.58893098 30.99555475 27.68803538 24.73345966]
 [ 1. 9.03727336 2.75430972 8.808669 81.67230985 24.89144991
 79.60634972 7.58622206 24.26180269 77.59264954]]
```