

# 3+Kmeans

2019 年 3 月 9 日

## 1 Kmeans 聚类

### 1.1 一、概念

Kmeans 是经典的无监督学习算法，不需要因变量  $Y$  来对数据进行学习，并根据学习结果将需要分类的数据分入之前分好的类中。

Kmeans 主要有以下几步：

1. 随即选取  $K$  个样本点作为中心点
2. 计算所有样本点到各个中心点的距离
3. 将样本点就近分入属于最近的中心那一类中，得到  $K$  类
4. 计算各类的中心

不断重复迭代上述几步，直到中心点不再变动，模型就训练完成了。

### 1.2 二、实施细节

#### 1.2.1 1. 距离的选取

距离公式的选取有多种，最常见的就是  $L1$  和  $L2$ 。

$L1$  距离又称曼哈顿距离，两点  $A(x_1, y_1)$ ,  $B(x_2, y_2)$  的  $L1$  距离为：

$$L1(A, B) = |x_1 - x_2| + |y_1 - y_2|$$

$L2$  距离又称欧式距离，两点  $A(x_1, y_1)$ ,  $B(x_2, y_2)$  的  $L2$  距离为：

$$L2(A, B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$L1$  和  $L2$  距离有更泛化的定义，即为闵科夫斯基 (Minkowski) 距离，对于  $A(x_1, x_2, \dots, x_n)$ ,  $B(y_1, y_2, \dots, y_n)$

$$Minkowski(A, B; p) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

也有其他的距离公式可以选取，根据实际情况选用不同的距离公式即可。

### 1.2.2 2. 新样本中心的计算

将样本点就近分类之后需要计算新的样本点进行迭代，新样本点的计算大多是对该类的样本点取均值得到

### 1.2.3 3. 损失函数

损失函数定义为各个样本点到其所属类的中心点的距离和，此处的距离和先前定义的距离保持一致，设有  $k$  类， $x_{ij}$  表示从属于第  $i$  类的第  $j$  个点，第  $i$  类有  $n_i$  个样本点，各类中心点为  $\mu_i$ ：

$$loss = \sum_{i=1}^l \sum_{j=1}^{n_i} distance(x_{ij}, \mu_i)$$

损失函数理论上会随着设定的中心点（类的数量）增加而不断降低，当只有一个类时损失最高，当所有样本点都是一个类时损失为 0。然而相同类数并不能保证收敛之后有相同的损失和中心点，所以在极端情况下，类数较多时的损失仍有可能比类数较少时的损失大。

### 1.2.4 4.K 的选取

K 的选取有多种方法，最经典直观的是肘部原则：当类数不断增加时，损失整体而言随之减少。因此可以重复计算类数为 2, 3, 4, ... 时的损失，取其平均然后作图，当前期 K 的增加会导致损失急剧地减小，函数图像会十分陡峭；当 K 达到一定数目之后，损失减小变得平缓，形成一个类似于肘部的形状，肘部的“关节”对应的 K 值即是合适的 K 值

## 1.3 三、应用

整个 Kmeans 算法在数学上并不复杂，但是实施起来有一定的繁琐程度，适合将算法分解成各个函数，以 pandas 的索引作为类的标识。

```
In [60]: import numpy as np
import pandas as pd
```

```
iris = pd.read_csv('Iris.data', header=None, names = ['sepal.l', 'sepal.w', 'petal.l', 'petal.w'])
iris.index = iris['class']
iris = iris.drop('class', axis=1)
iris.head(5)
```

```
Out[60]:
```

	sepal.l	sepal.w	petal.l	petal.w
class				
Iris-setosa	5.1	3.5	1.4	0.2
Iris-setosa	4.9	3.0	1.4	0.2
Iris-setosa	4.7	3.2	1.3	0.2
Iris-setosa	4.6	3.1	1.5	0.2
Iris-setosa	5.0	3.6	1.4	0.2

```
In [53]: def distance(a, b, p=2, dim=0):
          diff = np.abs(a-b)
          diff = np.power(diff, p)
          diff = np.power(np.sum(diff, axis=dim), 1/p)
          return diff

          a = np.array([0, 0, 0, 0])
          b = np.array([1, 1, 1, 1])
          print(distance(a, b))

          distance(a, iris, dim=1)
```

2.0

```
Out[53]: 1      6.345077
          1      5.916925
          1      5.836095
          1      5.749783
          1      6.321392
          1      6.886218
          1      5.896609
          1      6.232977
          1      5.456189
          1      5.989992
          1      6.718631
          1      6.099180
          1      5.831809
          1      5.358171
          1      7.149825
          1      7.366139
          1      6.798529
          1      6.349016
          1      7.064701
          1      6.541407
          1      6.606815
          1      6.489222
          1      5.929587
          1      6.327717
          1      6.184658
          1      6.049793
```

```
1      6.267376
1      6.448256
1      6.371813
1      5.910161
...
2      9.779059
0      8.198171
2     10.771258
0      8.615683
2      9.627045
2     10.065784
0      8.518216
0      8.570881
2      9.196195
2      9.850888
2     10.169562
2     11.036757
2      9.219544
0      8.705745
2      8.791473
2     10.525683
2      9.400532
2      9.168424
0      8.442748
2      9.528379
2      9.571834
2      9.408507
0      8.399405
2      9.827512
2      9.722140
2      9.285473
0      8.634234
2      9.071384
2      9.189668
0      8.547514
Length: 150, dtype: float64
```

是否收敛可以通过损失减少是否小于的阈值，或中心点是否移动判断。

一般而言二者均可，通过判断中心点是否移动更稳健（也更麻烦）。下面写出如何判断中心点是否移动的函数，但稍后不会用在训练的函数中。

因为计算机的特性，导致通过计算得来的两个浮点数几乎不可能相等，譬如一个不可逆矩阵的行列式可能不等于零（但实际上就是 0），为解决这种问题通常是设置一定的阈值，低于这个阈值就视为两个值相同，通常的阈值是  $1e-16$ ，但为了简便，下面的函数的阈值设置为 0.0001，以判断前后的中心点是否不同。

```
In [4]: def judge(a, b):
        diff = np.abs(a - b)
        diff = np.sum(diff)
        return diff < 0.0001

a = np.array([0, 0])
b = np.array([1, 1])
print('a : '+str(a)+' b : '+str(b))
print('Is a and b the same point? : '+str(judge(a, b)))

a = np.random.randn(2)
b = a.astype('float32')
print('a : '+str(a)+' b : '+str(b))
print('Is a and b the same point numerically? : '+str(a is b))
print('Is a and b the same point by distance? : '+str(judge(a, b)))

a : [0 0] b : [1 1]
Is a and b the same point? : False
a : [0.23559094 0.59014251] b : [0.23559093 0.5901425 ]
Is a and b the same point numerically? : False
Is a and b the same point by distance? : True
```

```
In [57]: def distance_overall(data, center):
        k = center.shape[0]
        dis = pd.DataFrame()
        for i in range(k):
            dis[str(i)] = distance(center.iloc[i], data, dim=1)

        dis = np.array(dis)
        index = np.argmin(dis, axis=1)
        dis = np.min(dis, axis=1)
        loss = np.sum(dis)
        return index, loss

np.random.seed(2099)
```

```

index = np.random.permutation(150)[0:3]
center = iris.iloc[index[0:3]]
distance_overall(iris, center)

```

```

Out[57]: (array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 0, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1,
                1, 1, 0, 1, 1, 2, 1, 1, 1, 1, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1,
                0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0], dtype=int64),
153.58512483193363)

```

```

In [62]: def kmeans(data, k=3):
    decay = 1
    loss_pre = 0
    loss_aft = float('inf')

    initial_index = np.random.permutation(data.shape[0])
    center = data.iloc[initial_index[0:k]]

    while decay > 0.01:
        tmp = np.array([0, 0, 0, 0])

        loss_pre = loss_aft
        data.index, loss_aft = distance_overall(data, center)
        for i in range(k):
            cluster = np.array(data.loc[i])
            tmp = np.vstack([tmp, np.mean(cluster, axis = 0)])

        center = pd.DataFrame(tmp)
        center.columns = ['sepal.l', 'sepal.w', 'petal.l', 'petal.w']
        center = center.drop(0, axis=0)
        decay = loss_pre-loss_aft

    return center, data, loss_aft

kmeans(iris)

```

```

Out[62]: (   sepal.l   sepal.w   petal.l   petal.w
1  6.850000  3.073684  5.742105  2.071053

```

```

2  5.006000  3.428000  1.462000  0.246000
3  5.901613  2.748387  4.393548  1.433871,
    sepal.l  sepal.w  petal.l  petal.w
1      5.1      3.5      1.4      0.2
1      4.9      3.0      1.4      0.2
1      4.7      3.2      1.3      0.2
1      4.6      3.1      1.5      0.2
1      5.0      3.6      1.4      0.2
1      5.4      3.9      1.7      0.4
1      4.6      3.4      1.4      0.3
1      5.0      3.4      1.5      0.2
1      4.4      2.9      1.4      0.2
1      4.9      3.1      1.5      0.1
1      5.4      3.7      1.5      0.2
1      4.8      3.4      1.6      0.2
1      4.8      3.0      1.4      0.1
1      4.3      3.0      1.1      0.1
1      5.8      4.0      1.2      0.2
1      5.7      4.4      1.5      0.4
1      5.4      3.9      1.3      0.4
1      5.1      3.5      1.4      0.3
1      5.7      3.8      1.7      0.3
1      5.1      3.8      1.5      0.3
1      5.4      3.4      1.7      0.2
1      5.1      3.7      1.5      0.4
1      4.6      3.6      1.0      0.2
1      5.1      3.3      1.7      0.5
1      4.8      3.4      1.9      0.2
1      5.0      3.0      1.6      0.2
1      5.0      3.4      1.6      0.4
1      5.2      3.5      1.5      0.2
1      5.2      3.4      1.4      0.2
1      4.7      3.2      1.6      0.2
..      ...      ...      ...      ...
0      6.9      3.2      5.7      2.3
2      5.6      2.8      4.9      2.0
0      7.7      2.8      6.7      2.0
2      6.3      2.7      4.9      1.8
0      6.7      3.3      5.7      2.1

```

0	7.2	3.2	6.0	1.8
2	6.2	2.8	4.8	1.8
2	6.1	3.0	4.9	1.8
0	6.4	2.8	5.6	2.1
0	7.2	3.0	5.8	1.6
0	7.4	2.8	6.1	1.9
0	7.9	3.8	6.4	2.0
0	6.4	2.8	5.6	2.2
2	6.3	2.8	5.1	1.5
0	6.1	2.6	5.6	1.4
0	7.7	3.0	6.1	2.3
0	6.3	3.4	5.6	2.4
0	6.4	3.1	5.5	1.8
2	6.0	3.0	4.8	1.8
0	6.9	3.1	5.4	2.1
0	6.7	3.1	5.6	2.4
0	6.9	3.1	5.1	2.3
2	5.8	2.7	5.1	1.9
0	6.8	3.2	5.9	2.3
0	6.7	3.3	5.7	2.5
0	6.7	3.0	5.2	2.3
2	6.3	2.5	5.0	1.9
0	6.5	3.0	5.2	2.0
0	6.2	3.4	5.4	2.3
2	5.9	3.0	5.1	1.8

```
[150 rows x 4 columns],  
97.20457357401651)
```

```
In [ ]:
```