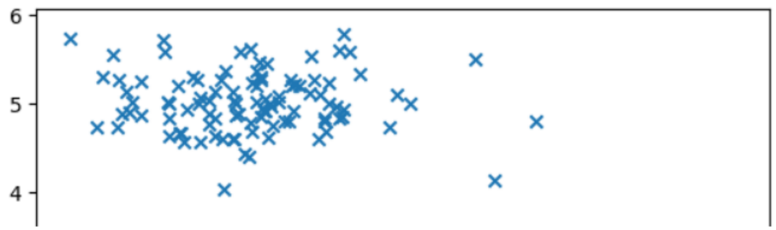
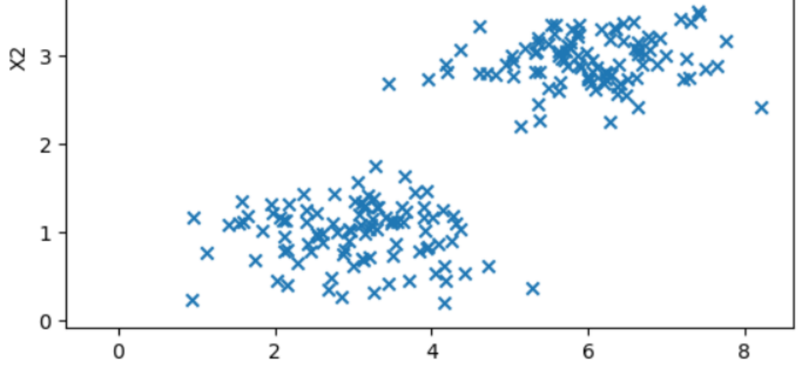
第10章 聚类

一、测试数据

[1]

```python
#load data
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt

data = sio.loadmat('ex9data2.mat')
X = data['X']
print(X.shape)
print(X[:5])


plt.scatter(X[:, 0], X[:, 1], marker='x')

plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```





二、K-Means

1. 寻找最近中心

[2]

```
import numpy as np

#计算距离
def computeDistance(A, B):
    return np.sqrt(np.sum(np.square(A-B)))

#为数据集x找到最近的质心的索引
def findClosestCentroids(X, centroids):
    k = centroids.shape[0]
    m = X.shape[0]
    idx = np.zeros((X.shape[0],1))
    for i in range(m):
        minDist = np.inf
        minIndex = -1
        for j in range(K):
            distance = computeDistance(X[i,:],centroids[j,:])
            if distance<minDist:
                minDist = distance
                minIndex = j
```

```python
            idx[i,:] = minIndex
    return idx

#用初始值测试一下
K = 3
initial_centroids = np.array([[3,3], [6,2], [8,5]])
idx = findClosestCentroids(X, initial_centroids)
print("should be 1, 3, 2\n", idx[:3])
```

```
should be 1, 3, 2
 [[0.]
 [2.]
 [1.]]
```

2. 重新计算中心

[3]

```python
#compute the mean of the data ,it is centroids
def change_centroids(X, idx, K):
    m, n = X.shape
    centroids = np.zeros((K, n))
    for i in range(K):
        index = np.where(idx.ravel() == i)
        centroids[i] = np.mean(X[index], axis=0)
    return centroids

#测试一下
print("Centroids computed after initial finding of closest centroids: \n")
print('the centroids should be\n[ 2.428301 3.157924 ]')
print('[ 5.813503 2.633656 ]\n[ 7.119387 3.616684 ]\n')

centroids = change_centroids(X, idx, K)
print(centroids)
```

```
Centroids computed after initial finding of closest centroids:

the centroids should be
[ 2.428301 3.157924 ]
```

```
[ 5.813503 2.633656 ]
[ 7.119387 3.616684 ]

[[2.42830111 3.15792418]
 [5.81350331 2.63365645]
 [7.11938687 3.6166844 ]]
```

3. 初始化中心

[4]

```python
#我们一开始应该随机初始化centroids
def initCentroids(X, K=3):
    m, n = X.shape
    centroids = np.zeros((K, n))
    randIndex = np.random.choice(m, K)
    centroids = X[randIndex]
    return centroids
```

4. K-Means Class

[5]

```python
import os
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt


class KMeans:
    def __init__(self, n_cluster=3, epochs=30, tolerance=1e-5):
        self.n_cluster = n_cluster
        self.epochs = epochs
        self.tolerance = tolerance

    # 计算距离
```

```python
    def compute_distance(self, A, B):
        return np.sqrt(np.sum(np.square(A-B)))

    # 随机初始化中心（KMeans）
    def rand_centroids(self, X):
        m, n = X.shape
        centroids = np.zeros((self.n_cluster, n))
        randIndex = np.random.choice(m, self.n_cluster)
        centroids = X[randIndex]
        # self.centroids = centroids
        return centroids

    # 随机初始化中心（KMeans++）
    # K-Means++算法在初始化聚类中心时的基本原则是使聚类中心之间的相互距离尽可能的远，其初始过程如下：
    #   1、在数据集中随机选择一个样本作为第一个初始化聚类中心；
    #   2、计算样本中每一个样本点与已经初始化的聚类中心的距离，并选择其中最短的距离；
    #   3、以概率选择距离最大的点作为新的聚类中心；
    #   4、重复2、3步直至选出k个聚类中心；
    #   5、对k个聚类中心使用K-Means算法计算最终的聚类结果。
    def rand_centroids_pp(self, X):
        # 定义内部函数：求样本到中心的最近距离
        def nearest(point, cluster_centers):
            '''
                计算point和cluster_centers之间的最小距离
                :param point: 当前的样本点
                :param cluster_centers: 当前已经初始化的聚类中心
                :return: 返回point与当前聚类中心的最短距离
            '''
            min_dist = float('inf')
            # 当前已经初始化聚类中心的个数
            m = np.shape(cluster_centers)[0]
            for i in range(m):
                # 计算point与每个聚类中心之间的距离
                d = self.compute_distance(point, cluster_centers[i, ])
                # 选择最短距离
                if min_dist > d:
                    min_dist = d
            return min_dist
```

```
        # 开始寻找中心
        # 初始化
        m, n = np.shape(X)
        centroids = np.zeros((self.n_cluster, n))

        # 1、随机选择一个样本点作为第一个聚类中心
        index = np.random.randint(0, m)
        centroids[0, ] = np.copy(X[index, ])

        # 2、初始化一个距离序列
        d = [0.0 for _ in range(m)]

        for k in range(1, self.n_cluster):
            sum_all = 0
            for j in range(m):
                # 3、对每一个样本找到最近的聚类中心点
                d[j] = nearest(X[j, ], centroids[0:k, ])
                # 4、将所有的最短距离相加
                sum_all += d[j]
            # 5、取得sum_all之间的随机值
            sum_all *= np.random.rand()
            # 6、获得距离最远的样本点作为聚类中心点
            # enumerate()函数用于将一个可遍历的数据对象（如列表、元组或字符串）组合为一个索引序列，同事列出数据和数据
下标一般用在for循环中
            for j, di in enumerate(d):
                sum_all -= di
                if sum_all > 0:
                    continue
                centroids[k] = np.copy(X[j, ])
                break
        # self.centroids = centroids
        return centroids

    # 中心更新
    def update_centroids(self, X):
        predict_class = self.predict(X)
        # update centroids
        centroids = self.centroids
```

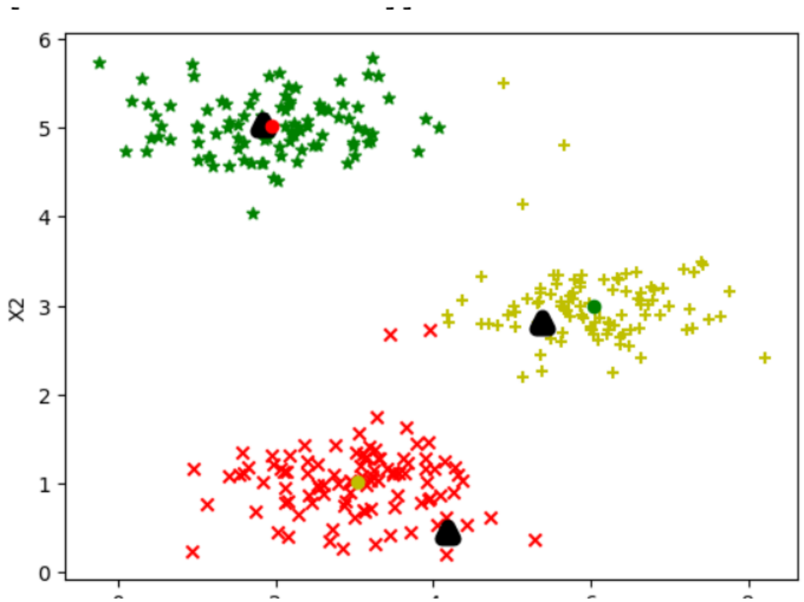
```python
        tolerance = 0.0
        for ct in range(len(centroids)):
            idx, = np.where(predict_class == ct)
            samples = X[idx, :]
            assert len(samples) > 0
            centroids[ct] = np.mean(samples, axis=0)
            tolerance += np.sqrt(np.sum(np.square(samples-centroids[ct])))
        self.centroids = centroids
        return centroids, tolerance

    # 训练
    def fit(self, X):
        self.centroids = self.rand_centroids_pp(X)
        tolerance_last = float('inf')
        tolerance_current = 0.0
        epoch = 0
        centroids_list = []
        while epoch < self.epochs and abs(tolerance_last-tolerance_current) > self.tolerance:
            tolerance_last = tolerance_current
            centroids, tolerance_current = self.update_centroids(X)
            self.centroids = centroids
            centroids_list.append(centroids)
            # print("tolerance={}", tolerance_current)
        return centroids, centroids_list

    # 预测
    def predict(self, X):
        # 初始化
        predict_class = np.zeros(shape=(len(X),))
        centroids = self.centroids
        for n_sample, arr in enumerate(X):
            min_distance = float("inf")
            p_class = 0
            for center in range(len(centroids)):
                distance = self.compute_distance(arr, centroids[center])
                if distance < min_distance:
                    min_distance = distance
                    p_class = center
```

```python
            predict_class[n_sample] = p_class
        return predict_class
```

[6]

```python
# 加载数据
path = os.path.abspath(os.path.dirname(__file__))
data = sio.loadmat(path + os.path.sep + 'ex9data2.mat')
X = data['X']
# print(X.shape)
# print(X[:5])
# plt.scatter(X[:, 0], X[:, 1], marker='x')
plt.xlabel('X1')
plt.ylabel('X2')
# plt.show()
# 用初始化的centroids测试聚类的过程和结果
kmeans = KMeans(n_cluster=3, epochs=100, tolerance=1e-9)
initial_centroids = kmeans.rand_centroids_pp(X)
print(initial_centroids)
# 训练数据并用不同的颜色画出分类中心
centroids, centroids_list = kmeans.fit(X)
print(centroids)
print(len(centroids_list))
plt.plot(centroids[0, 0], centroids[0, 1], "y-o")
plt.plot(centroids[1, 0], centroids[1, 1], "r-o")
plt.plot(centroids[2, 0], centroids[2, 1], "g-o")
# 用不同的颜色画出分类结果
predict_class = kmeans.predict(X)
idx0, = np.where(predict_class.ravel() == 0)
idx1, = np.where(predict_class.ravel() == 1)
idx2, = np.where(predict_class.ravel() == 2)
plt.scatter(X[idx0, 0], X[idx0, 1], marker='x', color='r')
plt.scatter(X[idx1, 0], X[idx1, 1], marker='*', color='g')
plt.scatter(X[idx2, 0], X[idx2, 1], marker='+', color='y')
for centroid in centroids_list:
    # print(centroid)
    plt.scatter(centroid[:, 0], centroid[:, 1], marker='o', color='blue')
plt.scatter(initial_centroids[:, 0], initial_centroids[:,
            1], marker='^', color='black', linewidths=7)
```

```
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



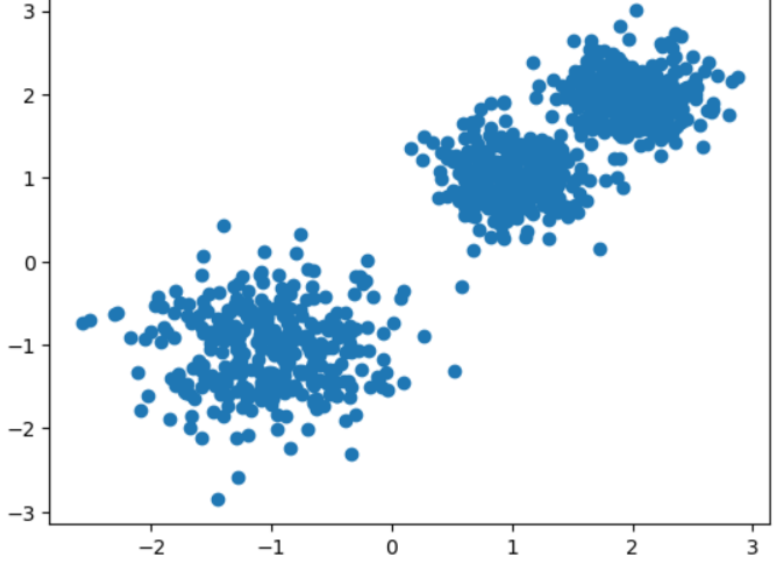
三、SKLean中K-Means算法

[-]

```
### 1. 普通k-Means
```

[7]

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# X为样本特征，y为样本簇类别，共1000个样本，每个样本2个特征，对应x和y轴，共4个簇，
# 簇中心在[-1,-1], [0,0],[1,1], [2,2]， 簇方差分别为[0.5, 0.3, 0.3, 0.3]
X, y = make_blobs(n_samples=1000, n_features=2, centers=[[-1, -1], [1, 1], [2, 2]], cluster_std=[0.5, 0.3,
0.3],random_state=12)
# 假设暂不知道y类别，不设置c=y，使用kmeans聚类
plt.scatter(X[:, 0], X[:, 1], marker='o')
plt.show()
```



[8]

```python
kmean_clf = KMeans(n_clusters=3, random_state=12)
kmean_clf.fit(X)
```

```
KMeans(n_clusters=3, random_state=12)
```

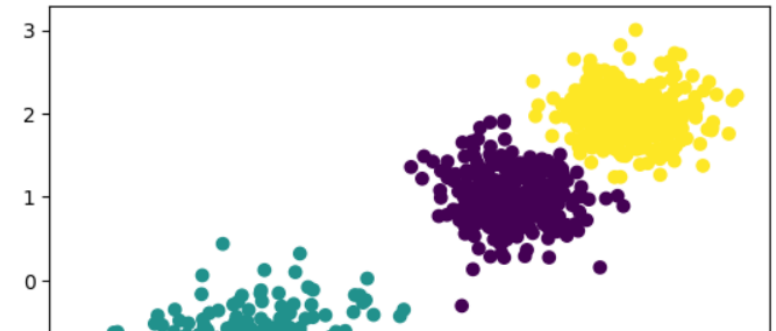
[10]

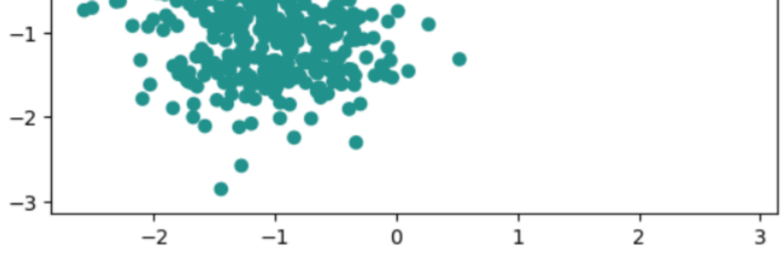
```python
y_pred = kmean_clf.predict(X)
print(kmean_clf)
```

```
KMeans(n_clusters=3, random_state=12)
```

[11]

```python
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.show()
```

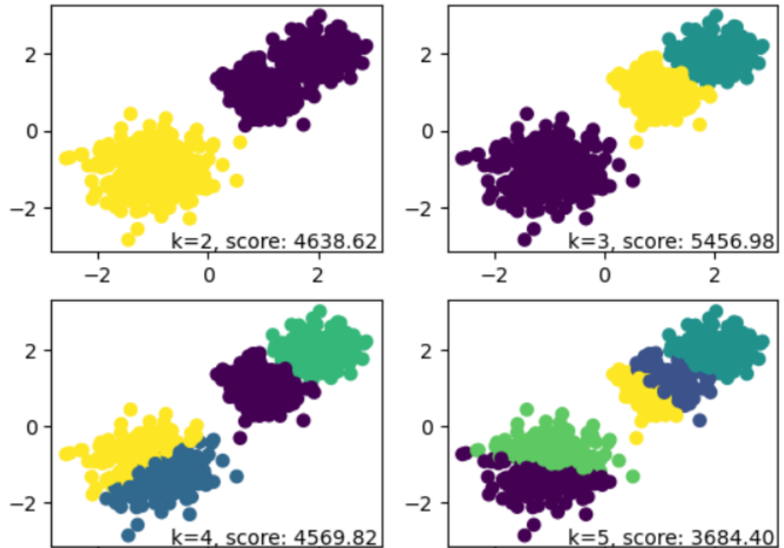




2. MiniBatchKMeans

[13]

```
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import calinski_harabasz_score

for index, k in enumerate((2, 3, 4, 5)):
    plt.subplot(2, 2, index + 1)
    mbatch_kmean = MiniBatchKMeans(n_clusters=k, batch_size=200, random_state=12)
    y_pred = mbatch_kmean.fit_predict(X)
    score = calinski_harabasz_score(X, y_pred)
    plt.scatter(X[:, 0], X[:, 1], c=y_pred)
    plt.text(.99, .01, ('k=%d, score: %.2f' % (k, score)), transform=plt.gca().transAxes, size=10,
             horizontalalignment='right')
plt.show()
```



参考：

- https://zhuanlan.zhihu.com/p/263056984 （聚类算法kmeans及kmeans++介绍）
- https://www.cnblogs.com/ahu-lichang/p/7161613.html （K-means聚类算法）
- https://blog.csdn.net/xc_zhou/article/details/88247783 （python实现K-Means算法）
- https://blog.csdn.net/gdkyxy2013/article/details/88381120 （Python实现K-Means++聚类算法）
- https://blog.csdn.net/zouxy09/article/details/17589329 （k均值聚类（k-means））