KMeans算法的代码实现

1.读取数据

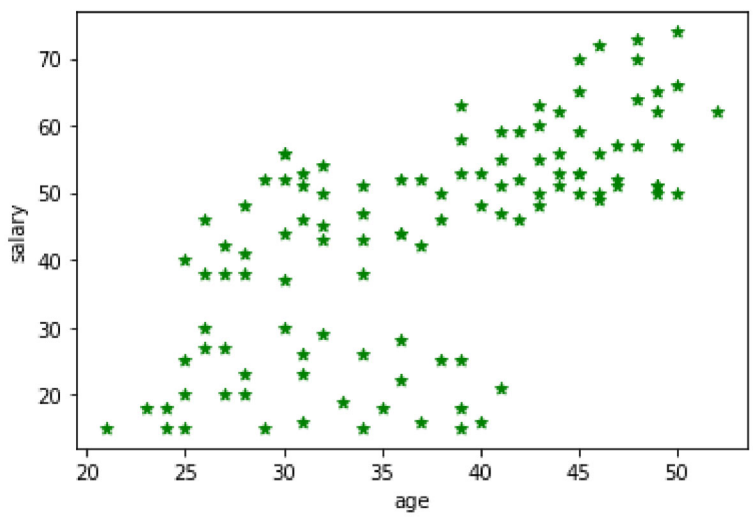
In [42]:
```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
data = pd.read_excel('KMeans_data.xlsx')
data.head()
```

Out[42]:

	年龄(岁)	收入(万元)
0	50	66
1	44	51
2	30	56
3	46	50
4	32	50

2.可视化展示

In [43]:
```
import matplotlib.pyplot as plt
plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c="green", marker='*')  # 以绿色星星样式绘制散点图
plt.xlabel('age')
plt.ylabel('salary')
plt.show()
```



3.数据建模

In [44]:
```
from sklearn.cluster import KMeans
kms = KMeans(n_clusters=3, random_state=53)
kms.fit(data)
label = kms.labels_
label = kms.fit_predict(data)
```

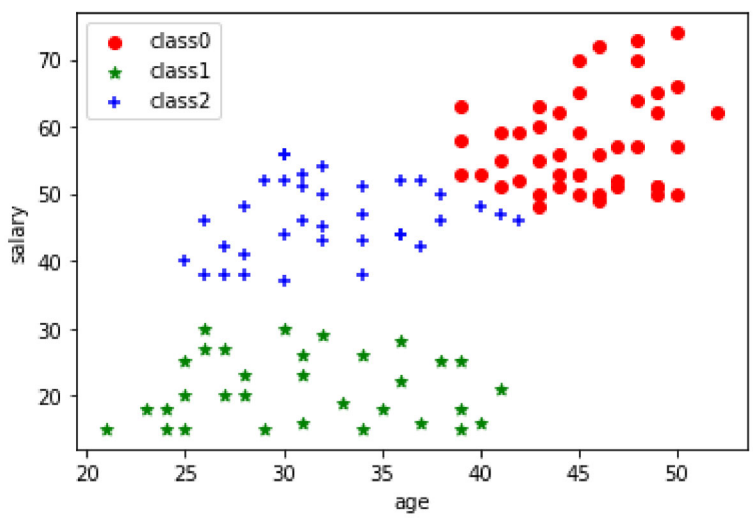
In [45]:
```
print(label)
```

```
[0 0 2 0 2 2 0 2 2 0 0 0 0 2 0 0 0 2 0 0 0 2 2 0 0 0 0 2 2 0 2 0 2 2 2 1 2
 0 2 1 0 0 2 0 2 0 2 0 0 2 2 1 0 2 0 0 0 0 2 0 2 2 2 2 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 2 0 2 1 1 1 1 1 1
 2]
```

4.建模效果可视化展示

In [46]:
```
# 以红色圆圈样式绘制类别0，以绿色星星样式绘制类别1，以蓝色加号样式绘制类别2
plt.scatter(data[label == 0].iloc[:, 0], data[label == 0].iloc[:, 1], c="red", marker='o', label='class0')
plt.scatter(data[label == 1].iloc[:, 0], data[label == 1].iloc[:, 1], c="green", marker='*', label='class1')
plt.scatter(data[label == 2].iloc[:, 0], data[label == 2].iloc[:, 1], c="blue", marker='+', label='class2')
plt.xlabel('age')  # 添加x轴名称
plt.ylabel('salary')  # 添加y轴名称
plt.legend()  # 设置图例
```

Out[46]: <matplotlib.legend.Legend at 0x1e5a4ad00a0>



5.查看各标签人的收入均值

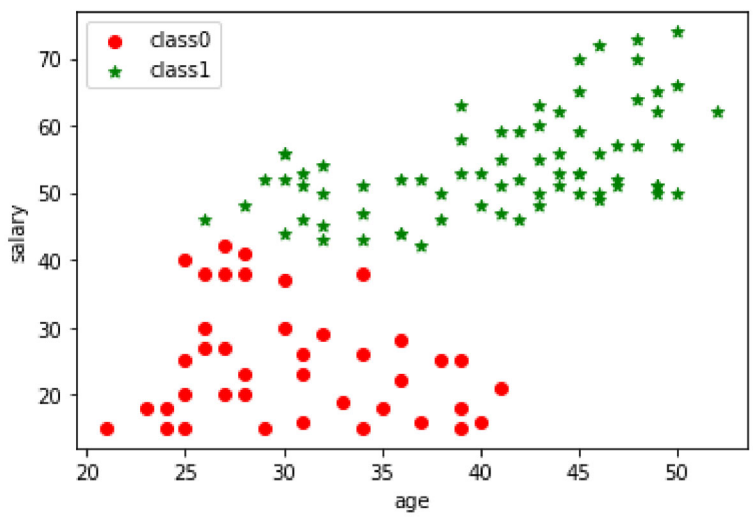
In [47]:
```
print(data[label == 0].iloc[:, 1].mean())  # 看下分类为标签0的人的收入均值，iloc[:, 1]为data表格的第二列，也即“收入”列
print(data[label == 1].iloc[:, 1].mean())
print(data[label == 2].iloc[:, 1].mean())
```

```
57.55555555555556
21.125
46.285714285714285
```

6.将数据聚为两类的可视化结果

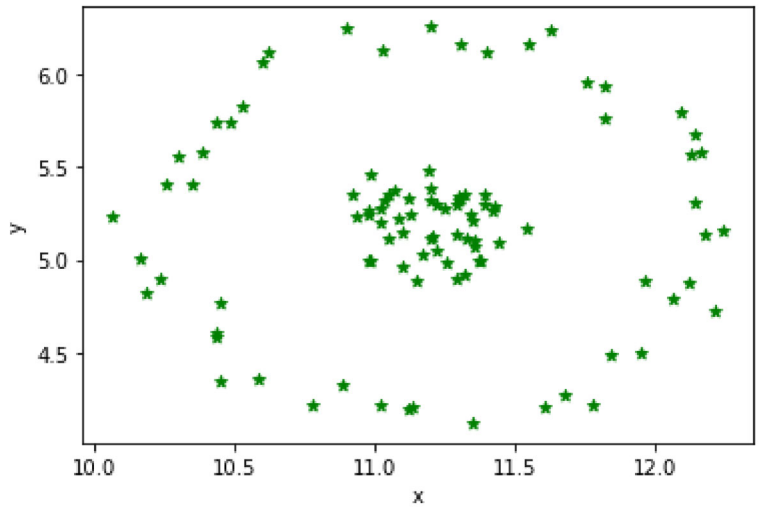
In [48]:
```
kms = KMeans(n_clusters=2, random_state=333)
kms.fit(data)
label = kms.labels_
label = kms.fit_predict(data)
plt.scatter(data[label == 0].iloc[:, 0], data[label == 0].iloc[:, 1], c="red", marker='o', label='class0')
plt.scatter(data[label == 1].iloc[:, 0], data[label == 1].iloc[:, 1], c="green", marker='*', label='class1')
plt.xlabel('age')
plt.ylabel('salary')
plt.legend()
```

Out[48]: <matplotlib.legend.Legend at 0x1e5aa1492e0>



DBSCAN算法的代码实现

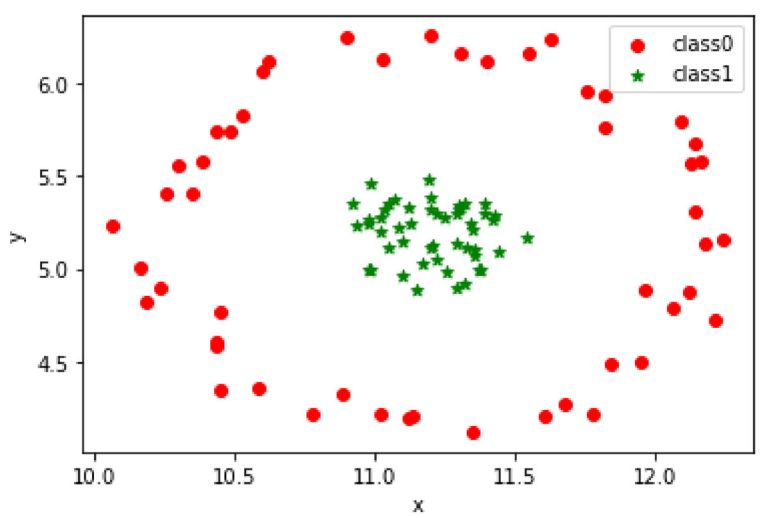
In [49]:
```
import pandas as pd
data = pd.read_excel('DBSCAN_data.xlsx')
import matplotlib.pyplot as plt
plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c="green", marker='*')  # 以绿色星星样式绘制散点图
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



In [50]:
```
from sklearn.cluster import DBSCAN
dbs = DBSCAN(eps=0.5, min_samples=3)
dbs.fit(data)
label_dbs = dbs.labels_
```

In [51]:
```
plt.scatter(data[label_dbs == 0].iloc[:, 0], data[label_dbs == 0].iloc[:, 1], c="red", marker='o', label='class0')
plt.scatter(data[label_dbs == 1].iloc[:, 0], data[label_dbs == 1].iloc[:, 1], c="green", marker='*', label='class1')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

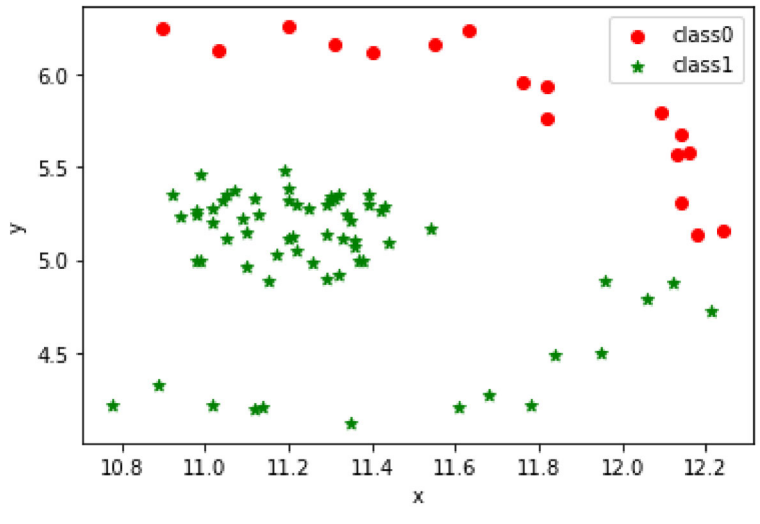
Out[51]: <matplotlib.legend.Legend at 0x1e5ab426610>



对比KMeans的实现效果

In [52]:
```
from sklearn.cluster import KMeans
KMs = KMeans(n_clusters=3)
KMs.fit(data)
label_kms = KMs.labels_
plt.scatter(data[label_kms == 0].iloc[:, 0], data[label_kms == 0].iloc[:, 1], c="red", marker='o', label='class0')  # 以红色圆圈
plt.scatter(data[label_kms == 1].iloc[:, 0], data[label_kms == 1].iloc[:, 1], c="green", marker='*', label='class1')  # 以绿色星
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
```

Out[52]: <matplotlib.legend.Legend at 0x1e5ab4250d0>



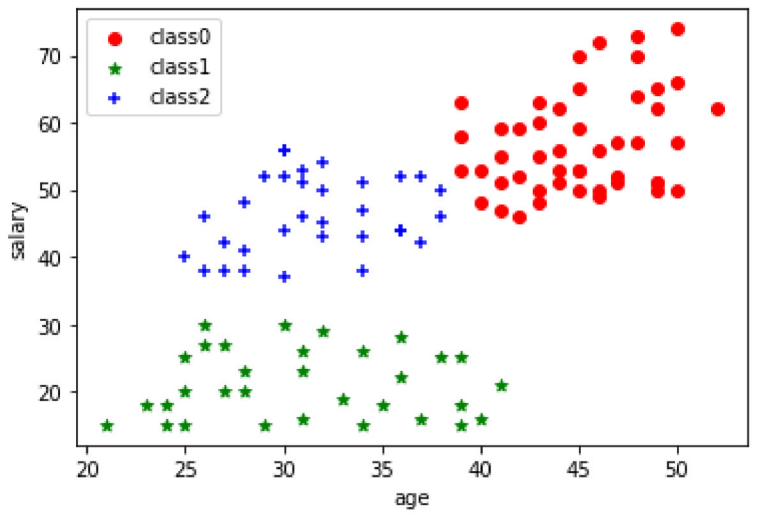
DBSCAN算法可视化：https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/ (https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/)

高斯混合聚类

In [53]:
```
from sklearn import mixture
data = pd.read_excel('KMeans_data.xlsx')
gmm = mixture.GaussianMixture(n_components=3)
gmm.fit(data)
label = gmm.predict(data)
```

In [54]:
```
plt.scatter(data[label == 0].iloc[:, 0], data[label == 0].iloc[:, 1], c="red", marker='o', label='class0')
plt.scatter(data[label == 1].iloc[:, 0], data[label == 1].iloc[:, 1], c="green", marker='*', label='class1')
plt.scatter(data[label == 2].iloc[:, 0], data[label == 2].iloc[:, 1], c="blue", marker='+', label='class2')
plt.xlabel('age')  # 添加x轴名称
plt.ylabel('salary')  # 添加y轴名称
plt.legend()  # 设置图例
```

Out[54]: <matplotlib.legend.Legend at 0x1e5ab536c70>

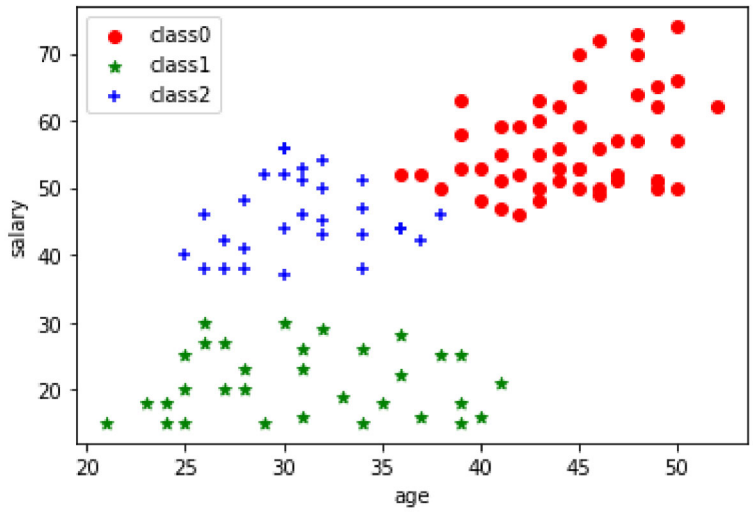


python层次聚类

In [55]:
```
from sklearn.cluster import AgglomerativeClustering

k=3
model=AgglomerativeClustering(n_clusters=k,linkage='ward')
model.fit(data)
label = model.labels_
```

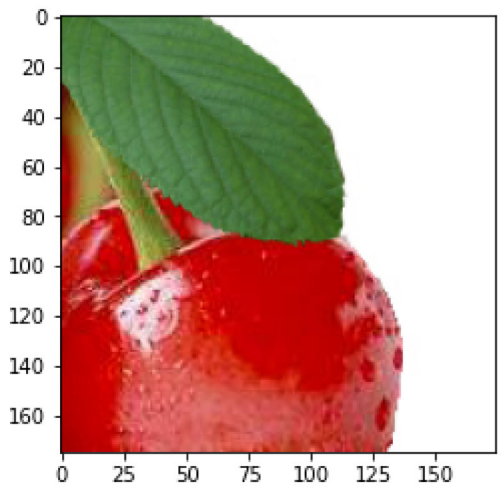
In [56]:
```
plt.scatter(data[label == 0].iloc[:, 0], data[label == 0].iloc[:, 1], c="red", marker='o', label='class0')
plt.scatter(data[label == 1].iloc[:, 0], data[label == 1].iloc[:, 1], c="green", marker='*', label='class1')
plt.scatter(data[label == 2].iloc[:, 0], data[label == 2].iloc[:, 1], c="blue", marker='+', label='class2')
plt.xlabel('age')  # 添加x轴名称
plt.ylabel('salary')  # 添加y轴名称
plt.legend()  # 设置图例
```

Out[56]: <matplotlib.legend.Legend at 0x1e5ab38a700>



KMeans实现聚类分割

In [57]:
```
import cv2
import numpy as np
img = cv2.imread('cherry.jpg')
plt.imshow(img[:,:,::-1])  # 转换rgb通道顺序再展示图像
plt.show()
```

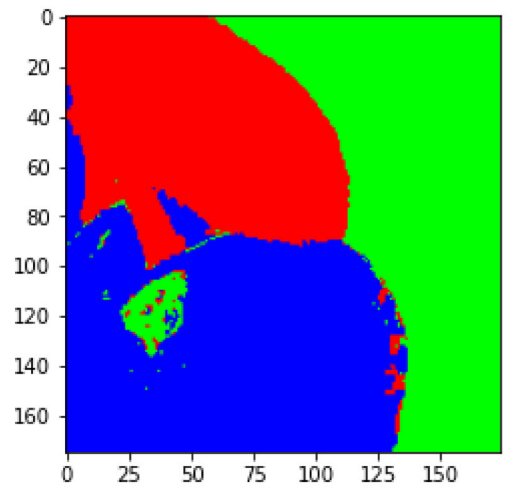


In [58]:

```
def get_feature(img):#获取图像各个点的特征
    row,col = img.shape[:2]
    features = []
    for i in range(0, row):
        for j in range(0, col):
            r = img[i, j, 2]
            g = img[i, j, 1]
            b = img[i, j, 0]
            features.append([r, g, b, i, j])
    features = np.array(features, 'f')
    return features

def distance(vecA, vecB, method='rgb',alpha=3.5):   #计算距离
    rgb_dis = np.sqrt(np.power(vecA[0] - vecB[0], 2) + np.power(vecA[1] - vecB[1], 2) + np.power(vecA[2] - vecB[2], 2))
    loc_dis =  np.sqrt(np.power(vecA[3] - vecB[3], 2) + np.power(vecA[4] - vecB[4], 2))
    if method == 'rgb':  #只使用颜色作为特征
        return rgb_dis
    else :  #method='rgb_loc', 使用颜色和坐标信息作为特征
        return rgb_dis + alpha*loc_dis

def sel_init_cen(features, k):  #随机选择K个初始聚类中心
    np.random.seed(0)
    rands = [(int)(np.random.random() * (features.shape[0])) for _ in range(k)]
    # 选取初始中心
    centors = [features[rands[i]] for i in range(k)]
    return centors

def get_centor(feature, centors, method='rgb',alpha=3.5): #迭代计算聚类中心
    k = len(centors)
    # 建立k个类别数据的空集合
    classes = [[] for _ in range(k)]

    # 设置大步长，减少计算时间
    for i in range(0, feature.shape[0],10):
        # node到k个聚类中心的距离
        dists = [distance(feature[i], centor, method) for centor in centors]
        # 判为距离最近的类别，并重新计算聚类中心(平均值)
        for j in range(k):
            if min(dists) == distance(feature[i], centors[j], method, alpha):
                classes[j].append(feature[i])
                break
    for j in range(k):
        centors[j] = np.mean(classes[j], axis=0)
    return centors

def image2k(img, centors, method='rgb'):  #根据聚类中心进行图像分类

    row,col = img.shape[:2]
    k = len(centors)
    # 定义颜色库
    colors = [ [255, 0, 0], [0, 255, 0], [0, 0, 255],
               [255, 255, 0], [0, 255, 255], [255, 0, 255]]
    for i in range(0, row):
        for j in range(0, col):
            # 当前像素到k个聚类中心的距离
            dists = [distance([img[i][j][2],img[i][j][1],img[i][j][0],i,j], centor, method) for centor in centors]
            for ks in range(k):
                if min(dists) == distance([img[i][j][2],img[i][j][1],img[i][j][0],i,j], centors[ks], method ):
                    img[i][j] = colors[ks % len(colors)]
    return img
```

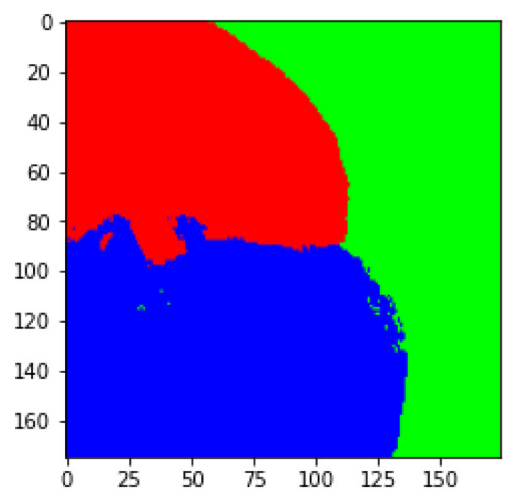
只使用颜色信息进行聚类分割

In [59]:
```
img = cv2.imread('cherry.jpg')
copy_img = img.copy()
k = 3
features = get_feature(img)
#获取k个随机初始聚类中心
init_cens = sel_init_cen(features, k)
centers = init_cens
#迭代计算K个聚类中心
for i in range(5):
    centers = get_centor(features, centers, method='rgb')
    # 显示k分类的图像
res_img = image2k(copy_img, centers, method='rgb')

plt.imshow(res_img)
plt.show()
```



使用颜色和坐标信息作为特征进行聚类分割

In [60]:
```
img = cv2.imread('cherry.jpg')
copy_img = img.copy()
k = 3
features = get_feature(img)
#获取k个随机初始聚类中心
init_cens = sel_init_cen(features, k)
centers = init_cens
#迭代计算K个聚类中心
for i in range(5):
    centers = get_centor(features, centers, method='rgb_loc')
    # 显示k分类的图像
res_img = image2k(copy_img, centers, method='rgb_loc')

plt.imshow(res_img)
plt.show()
```



In [ ]:

In [ ]: