


# K-means 计算 anchor boxes

hrsstudy 于 2017-05-04 18:25:09 发布

分类专栏: 机器学习 文章标签: k-means算法 YOLO

 机器学习 专栏收录该内容

1 订阅 13 篇文章

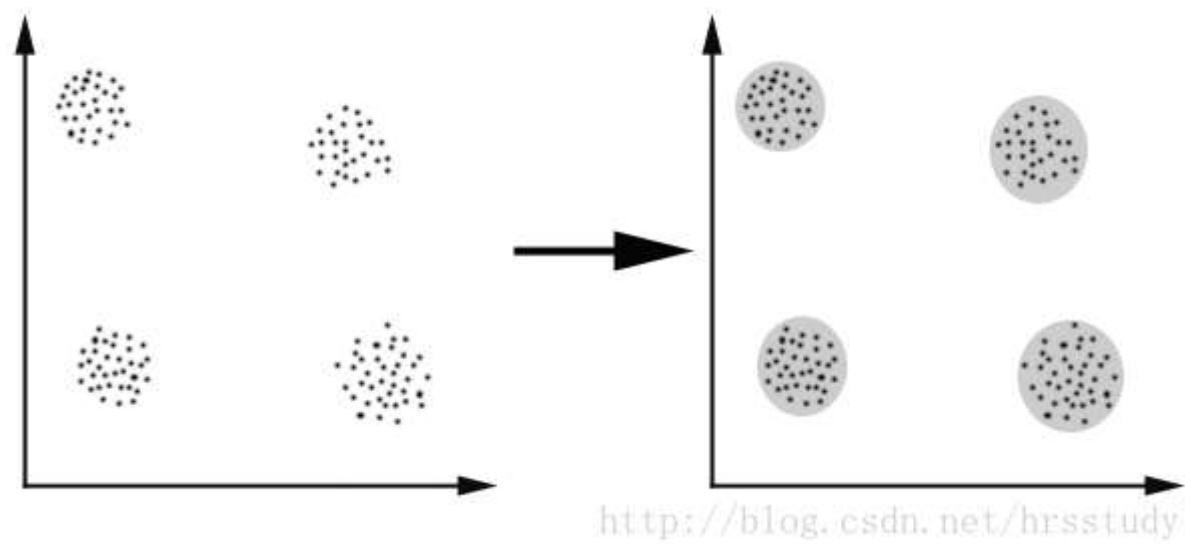
订阅专栏

## k-means 原理

K-means算法是很典型的基于距离的聚类算法，采用距离作为相似性的评价指标，即认为两个对象的距离越近，其相似度就越大。该算法认为簇是由距离靠近的对象组成的，因此把得到紧凑且独立的簇作为最终目标。

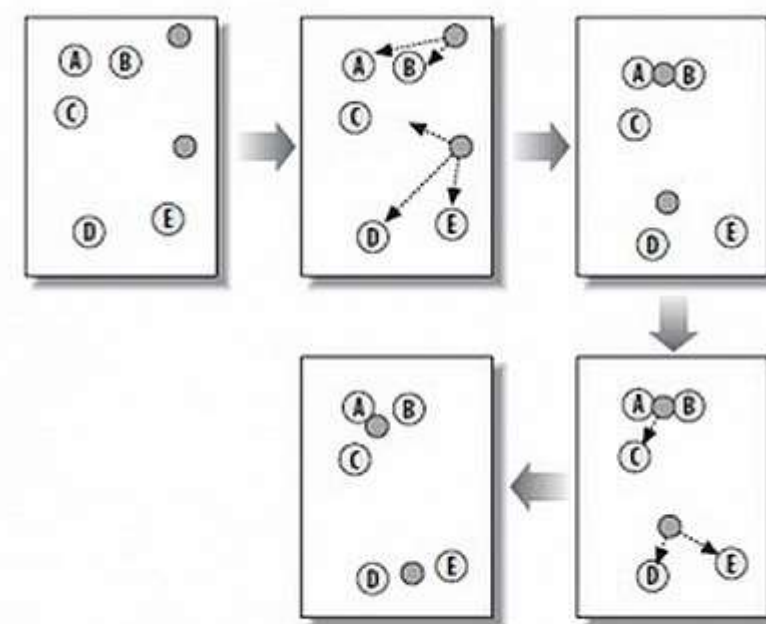
## 问题

K-Means算法主要解决的问题如下图所示。我们可以看到，在图的左边有一些点，我们用肉眼可以看出来有四个点群，K-Means算法被用来找出这几个点群。



## 算法概要





<http://blog.csdn.net/hrsstudy>

从上图中，我们可以看到，A, B, C, D, E 是五个在图中点。而灰色的点是我们的种子点，也就是我们用来找点群的点。有两个种子点，所以 $K=2$ 。

然后，K-Means的算法如下：

随机在图中取 $K$ （这里 $K=2$ ）个种子点。

然后对图中的所有点求到这 $K$ 个种子点的距离，假如点 $P_i$ 离种子点 $S_i$ 最近，那么 $P_i$ 属于 $S_i$ 点群。（上图中，我们可以看到A,B属于上面的种子点，C,D,E属于下面中部的种子点）

接下来，我们要移动种子点到属于他的“点群”的中心。（见图上的第三步）

然后重复第2) 和第3) 步，直到，种子点没有移动（我们可以看到图中的第四步上面的种子点聚合了A,B,C，下面的种子点聚合了D, E）。

### k-means算法缺点

- 1、需要提前指定 $k$
- 2、k-means算法对种子点的初始化非常敏感

### k-means++算法

k-means++是选择初始种子点的一种算法，其基本思想是：初始的聚类中心之间的相互距离要尽可能的远。

方法如下：

- 1.从输入的数据点集合中随机选择一个点作为第一个聚类中心
- 2.对于数据集中的每一个点 $x$ ，计算它与最近聚类中心(指已选择的聚类中心)的距离 $D(x)$
- 3.选择一个新的数据点作为新的聚类中心，选择的原则是： $D(x)$ 较大的点，被选取作为聚类中心的概率较大



- 4.重复2和3直到k个聚类中心被选出来
- 5.利用这k个初始的聚类中心来运行标准的k-means算法

第2、3步选择新点的方法如下：

- a.对于每个点，我们都计算其和最近的一个“种子点”的距离D(x)并保存在一个数组里，然后把这些距离加起来得到Sum(D(x))。
- b.然后，再取一个随机值，用权重的方式来取计算下一个“种子点”。这个算法的实现是，先用Sum(D(x))乘以随机值Random得到值r，然后用currSum += D(x)，直到其currSum>r，此时的点就是下一个“种子点”。原因见下图：



假设A、B、C、D的D(x)如上图所示，当算法取值Sum(D(x))\*random时，该值会以较大的概率落入D(x)较大的区间内，所以对应的点会以较大的概率被选中作为新的聚类中心。

### k-means 计算 anchor boxes

根据YOLOv2的论文，YOLOv2使用anchor boxes来预测bounding boxes的坐标。YOLOv2使用的anchor boxes和Faster R-CNN不同，不是手选的先验框，而是通过k-means得到的。YOLO的标记文件格式如下：

```
1 | <object-class> <x> <y> <width> <height>
```

object-class是类的索引，后面的4个值都是相对于整张图片的比例。  
x是ROI中心的x坐标，y是ROI中心的y坐标，width是ROI的宽，height是ROI的高。

卷积神经网络具有平移不变性，且anchor boxes的位置被每个栅格固定，因此我们只需要通过k-means计算出anchor boxes的width和height即可，即object-class,x,y三个值我们不需要。

由于从标记文件的width，height计算出的anchor boxes的width和height都是相对于整张图片的比例，而YOLOv2通过anchor boxes直接预测bounding boxes的坐标时，坐标是相对于栅格边长的比例（0到1之间），因此要将anchor boxes的width和height也转换为相对于栅格边长的比例。转换公式如下：

```
1 | w=anchor_width*input_width/downsamples
2 | h=anchor_height*input_height/downsamples
```

例如：

卷积神经网络的输入为416\*416时，YOLOv2网络的降采样倍率为32，假如k-means计算得到一个anchor box的anchor\_width=0.2，anchor\_height=0.6，则：

```
1 | w=0.2*416/32=0.2*13=2.6
2 | h=0.6*416/32=0.6*13=7.8
```



## 距离公式

因为使用欧氏距离会让大的bounding boxes比小的bounding boxes产生更多的error，而我們希望能通过anchor boxes获得好的IOU scores，并且IOU scores是与box的尺寸无关的。为此作者定义了新的距离公式：

```
1 | d(box,centroid)=1-IOU(box,centroid)
```

在计算anchor boxes时我们将所有boxes中心点的x, y坐标都置为0，这样所有的boxes都处在相同的位置上，方便我们通过新距离公式计算boxes之间的相似度。

## 代码实现

计算anchor boxes的python工具已上传至GitHub：

[https://github.com/PaulChongPeng/darknet/blob/master/tools/k\\_means\\_yolo.py](https://github.com/PaulChongPeng/darknet/blob/master/tools/k_means_yolo.py)

k\_means\_yolo.py代码如下：

```
1 | # coding=utf-8
2 | # k-means ++ for YOLOv2 anchors
3 | # 通过k-means ++ 算法获取YOLOv2需要的anchors的尺寸
4 | import numpy as np
5 |
6 | # 定义Box类，描述bounding box的坐标
7 | class Box():
8 |     def __init__(self, x, y, w, h):
9 |         self.x = x
10 |        self.y = y
11 |        self.w = w
12 |        self.h = h
13 |
14 |
15 | # 计算两个box在某个轴上的重叠部分
16 | # x1是box1的中心在该轴上的坐标
17 | # len1是box1在该轴上的长度
18 | # x2是box2的中心在该轴上的坐标
19 | # len2是box2在该轴上的长度
20 | # 返回值是该轴上重叠的长度
21 | def overlap(x1, len1, x2, len2):
22 |     len1_half = len1 / 2
23 |     len2_half = len2 / 2
24 |
25 |     left = max(x1 - len1_half, x2 - len2_half)
26 |     right = min(x1 + len1_half, x2 + len2_half)
27 |
28 |     return right - left
29 |
30 |
31 | # 计算box a 和box b 的交集面积
```



```
32 # a和b都是Box类型实例
33 # 返回值area是box a 和box b 的交集面积
34 def box_intersection(a, b):
35     w = overlap(a.x, a.w, b.x, b.w)
36     h = overlap(a.y, a.h, b.y, b.h)
37     if w < 0 or h < 0:
38         return 0
39
40     area = w * h
41     return area
42
43
44 # 计算 box a 和 box b 的并集面积
45 # a和b都是Box类型实例
46 # 返回值u是box a 和box b 的并集面积
47 def box_union(a, b):
48     i = box_intersection(a, b)
49     u = a.w * a.h + b.w * b.h - i
50     return u
51
52
53 # 计算 box a 和 box b 的 iou
54 # a和b都是Box类型实例
55 # 返回值是box a 和box b 的iou
56 def box_iou(a, b):
57     return box_intersection(a, b) / box_union(a, b)
58
59
60 # 使用k-means ++ 初始化 centroids, 减少随机初始化的centroids对最终结果的影响
61 # boxes是所有bounding boxes的Box对象列表
62 # n_anchors是k-means的k值
63 # 返回值centroids 是初始化的n_anchors个centroid
64 def init_centroids(boxes,n_anchors):
65     centroids = []
66     boxes_num = len(boxes)
67
68     centroid_index = np.random.choice(boxes_num, 1)
69     centroids.append(boxes[centroid_index])
70
71     print(centroids[0].w,centroids[0].h)
72
73     for centroid_index in range(0,n_anchors-1):
74
75         sum_distance = 0
76         distance_thresh = 0
77         distance_list = []
78         cur_sum = 0
79
80         for box in boxes:
81             min_distance = 1
82             for centroid_i, centroid in enumerate(centroids):
```



```

83         distance = (1 - box_iou(box, centroid))
84         if distance < min_distance:
85             min_distance = distance
86         sum_distance += min_distance
87         distance_list.append(min_distance)
88
89     distance_thresh = sum_distance*np.random.random()
90
91     for i in range(0,boxes_num):
92         cur_sum += distance_list[i]
93         if cur_sum > distance_thresh:
94             centroids.append(boxes[i])
95             print(boxes[i].w, boxes[i].h)
96             break
97
98     return centroids
99
100
101 # 进行 k-means 计算新的centroids
102 # boxes是所有bounding boxes的Box对象列表
103 # n_anchors是k-means的k值
104 # centroids是所有簇的中心
105 # 返回值new_centroids 是计算出的新簇中心
106 # 返回值groups是n_anchors个簇包含的boxes的列表
107 # 返回值loss是所有box距离所属的最近的centroid的距离的和
108 def do_kmeans(n_anchors, boxes, centroids):
109     loss = 0
110     groups = []
111     new_centroids = []
112     for i in range(n_anchors):
113         groups.append([])
114         new_centroids.append(Box(0, 0, 0, 0))
115
116     for box in boxes:
117         min_distance = 1
118         group_index = 0
119         for centroid_index, centroid in enumerate(centroids):
120             distance = (1 - box_iou(box, centroid))
121             if distance < min_distance:
122                 min_distance = distance
123                 group_index = centroid_index
124             groups[group_index].append(box)
125             loss += min_distance
126             new_centroids[group_index].w += box.w
127             new_centroids[group_index].h += box.h
128
129     for i in range(n_anchors):
130         new_centroids[i].w /= len(groups[i])
131         new_centroids[i].h /= len(groups[i])
132
133     return new_centroids, groups, loss

```



```

134
135
136 # 计算给定bounding boxes的n_anchors数量的centroids
137 # label_path是训练集列表文件地址
138 # n_anchors 是anchors的数量
139 # loss_convergence是允许的loss的最小变化值
140 # grid_size * grid_size 是栅格数量
141 # iterations_num是最大迭代次数
142 # plus = 1时启用k means ++ 初始化centroids
143 def compute_centroids(label_path,n_anchors,loss_convergence,grid_size,iterations_
144
145     boxes = []
146     label_files = []
147     f = open(label_path)
148     for line in f:
149         label_path = line.rstrip().replace('images', 'labels')
150         label_path = label_path.replace('JPEGImages', 'labels')
151         label_path = label_path.replace('.jpg', '.txt')
152         label_path = label_path.replace('.JPEG', '.txt')
153         label_files.append(label_path)
154     f.close()
155
156     for label_file in label_files:
157         f = open(label_file)
158         for line in f:
159             temp = line.strip().split(" ")
160             if len(temp) > 1:
161                 boxes.append(Box(0, 0, float(temp[3]), float(temp[4])))
162
163     if plus:
164         centroids = init_centroids(boxes, n_anchors)
165     else:
166         centroid_indices = np.random.choice(len(boxes), n_anchors)
167         centroids = []
168         for centroid_index in centroid_indices:
169             centroids.append(boxes[centroid_index])
170
171     # iterate k-means
172     centroids, groups, old_loss = do_kmeans(n_anchors, boxes, centroids)
173     iterations = 1
174     while (True):
175         centroids, groups, loss = do_kmeans(n_anchors, boxes, centroids)
176         iterations = iterations + 1
177         print("loss = %f" % loss)
178         if abs(old_loss - loss) < loss_convergence or iterations > iterations_num
179             break
180         old_loss = loss
181
182     for centroid in centroids:
183         print(centroid.w * grid_size, centroid.h * grid_size)
184

```



```
185         # print result
186         for centroid in centroids:
187             print("k-means result: \n")
188             print(centroid.w * grid_size, centroid.h * grid_size)
189
190
191 label_path = "/raid/pengchong_data/Data/Lists/paul_train.txt"
192 n_anchors = 5
193 loss_convergence = 1e-6
194 grid_size = 13
195 iterations_num = 100
196 plus = 0
197 compute_centroids(label_path,n_anchors,loss_convergence,grid_size,iterations_num,
```

## 参考

<http://coolshell.cn/articles/7779.html>

<http://www.cnblogs.com/shelocks/>

“相关推荐”对你有帮助么？



非常没帮助



没帮助



一般



有帮助



非常有帮助

[关于我们](#) [招贤纳士](#) [商务合作](#) [寻求报道](#) ☎ 400-660-0108 ✉ [kefu@csdn.net](mailto:kefu@csdn.net) 🗣 [在线客服](#) 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 [经营性网站备案信息](#) [北京互联网违法和不良信息举报中心](#)  
[家长监护](#) [网络110报警服务](#) [中国互联网举报中心](#) [Chrome商店下载](#) ©1999-2022北京创新乐知网络技术有限公司 [版权与免责声明](#) [版权申诉](#)  
[出版物许可证](#) [营业执照](#)

