k-均值(k-mean)

作者：刘伟杰 日期:2015-12-13
参考：
［1］《机器学习实战》 Peter

1. 理论

1. 概述：

   k − mean是一个用于聚类的方法。训练过程就是在特征空间中寻找最优的k个点，使得训练集每个点到这个k个点中最近“距离”的和最小，靠近同一个点的样本分为同一类。这个k个点称为这一类的质心，当输入新样本时就计算与各质心的“距离”，归类到最近的一类。

   k − mean的关键就是如何定义“距离”以及如何寻找最优的k个点。

2. 距离：

   欧式距离、皮尔逊距离、余弦距离…

3. 训练方法：

   训练方法就是搜索k个点的优化方法。先介绍k − mean常用的普通方法，再介绍改进的二分方法。

   - 普通方法：

     优点: 容易实现

     缺点: 容易陷入到局部最小值，在数据量大的时候收敛较慢

     伪代码：

     ```
     创建k个点作为起始的质心(经常是随机选择)
     当任意一个点的类分配结果发生改变时：
         对数据集中的每一个点：
             对每个质心：
                 计算质心与数据点之间的距离
             将数据点分配到距离其最近的类上
         对每一个类，计算类中所有点的均值作为新的质心
     ```

     python代码：

```
对应附录中的:
    def k_means(data_set, k, distance = distance_Eclud, rand_center = rand_cer
```

- 二分法:

  优点: 减小了陷入局部最优的可能性

  缺点: 算法复杂

  伪代码:

  ```
  将所有的点看成一个类
  当类的数目小于k时:
      对每一个类:
          计算总误差
          在给定的类上面进行2-均值聚类
          计算将该类一分为二后的总误差
      选择使得误差最小的哪个类进行划分
  ```

  python代码

  ```
  对应附录中的:
      def bi_k_means(data_set, k, distance = distance_Eclud, rand_center = rand_
  ```

附录:

scikit-learn:

```
#Import Library
from sklearn.cluster import KMeans
#Assumed you have, X (attributes) for training data set and x_test(attributes) of test
# Create KNeighbors classifier object model
k_means = KMeans(n_clusters=3, random_state=0)
# Train the model using the training sets and check score
model.fit(X)
#Predict Output
predicted= model.predict(x_test)
```

我的实现:

```python
# -*- coding: utf-8 -*-
'''
    this model is used for K-mean
    @author: Liu Weijie
'''

from numpy import *
import matplotlib.pyplot as plt


def load_data(filename):
    data_set = []
    with open(filename, 'r') as f:
        for line in f.readlines():
            cur_line = line.strip().split('\t')
            data_set.append([float(x) for x in cur_line])
    return data_set


def distance_Eclud(vec_A, vec_B):
    vec_A = mat(vec_A); vec_B = mat(vec_B)
    return sqrt(sum(power(vec_A - vec_B, 2)))


def rand_center(data_set, k):
    data_mat = mat(data_set)

    n = shape(data_mat)[1]
    rand_center = mat(zeros((k,n)))
    for j in range(n):
        min_j = min(data_mat[:,j])
        range_j = float(max(data_mat[:,j]) - min_j)
        rand_center[:,j] = min_j + range_j*random.rand(k,1)
    return rand_center


def k_means(data_set, k, distance = distance_Eclud, rand_center = rand_center):
    data_mat = mat(data_set);
    m, n = shape(data_mat)

    # 先随机生成中心
    center_list = rand_center(data_set, k)
    cluster_assment = mat(zeros((m,2)))

    is_cluster_changed = True
    while is_cluster_changed: # 如果簇有改变
        is_cluster_changed = False

        # 对所有的点分配簇
        for j in range(m):
            dist_min = inf; min_index = -1;
```

```
            for i in range(k):
                dist = distance(data_mat[j,:], center_list[i,:])
                if dist < dist_min:
                    dist_min = dist
                    min_index = i
            if min_index != cluster_assment[j,0]: is_cluster_changed = True # 如果簇分配
            cluster_assment[j,:] = min_index, dist_min**2

        # 计算新的中心
        for cent in range(k):
            pts_cluster = data_mat[nonzero(cluster_assment[:,0].A == cent)[0]]
            if shape(pts_cluster)[0] == 0: # 如果该中心没有分配的点，则重新随机生成该中心
                center_list[cent,:] = rand_center(data_set,1)[0,:]
            else:
                center_list[cent,:] = mean(pts_cluster, axis = 0)

    return center_list, cluster_assment


def test_k_means():
    data_set = load_data('testSet.txt')
    center_list, cluster_assment = k_means(data_set, 4)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(mat(data_set)[:,0].flatten().A[0],mat(data_set)[:,1].flatten().A[0], ma
    ax.scatter(mat(center_list)[:,0].flatten().A[0],mat(center_list)[:,1].flatten().A[
    plt.show()


def bi_k_means(data_set, k, distance = distance_Eclud, rand_center = rand_center):
    # 准备工作
    data_mat = mat(data_set)
    m, n = shape(data_mat)

    # 初始化簇和中心
    center_init = mean(data_mat, axis=0)
    center_list = [center_init.tolist()[0]]
    cluster_assment = mat(zeros((m,2)))
    for i in range(m):
        cluster_assment[i,1] = distance(center_list[0], data_mat[i, :])**2

    # while 分类数小于k
    while len(center_list) < k:
        # 寻找合适再次划分的簇
        lowest_SSE = inf
        for i in range(len(center_list)):
            tep_data_mat = data_mat[nonzero(cluster_assment[:,0].A == i)[0], :]
            new_center, new_cluster = k_means(tep_data_mat, 2, distance=distance, rand
            SSE_split = sum(new_cluster[:,1])
            SSE_nosplit = sum(cluster_assment[nonzero(cluster_assment[:,0].A != i)[0],
            if (SSE_split + SSE_nosplit) < lowest_SSE:
                best_split_index = i
```

```
                best_new_center = new_center
                best_new_cluster = new_cluster
                lowest_SSE = SSE_split + SSE_nosplit

        # 更新簇
        best_new_cluster[nonzero(best_new_cluster[:,0].A == 1)[0],0] = len(center_list
        best_new_cluster[nonzero(best_new_cluster[:,0].A == 0)[0],0] = best_split_inde
        cluster_assment[nonzero(cluster_assment[:,0].A == best_split_index),:] = best_
        center_list[best_split_index] = best_new_center[0].tolist()[0]
        center_list.append(best_new_center[1].tolist()[0])

        print 'center_list', center_list

    return center_list, cluster_assment


def test_bi_k_means():
    data_set = load_data('testSet.txt')
    center_list, cluster_assment = bi_k_means(data_set, 4)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(mat(data_set)[:,0].flatten().A[0],mat(data_set)[:,1].flatten().A[0], ma
    ax.scatter(mat(center_list)[:,0].flatten().A[0],mat(center_list)[:,1].flatten().A[
    plt.show()

if __name__ == '__main__':
    test_bi_k_means()
```