【华子】机器学习020-使用K-means算法对数据进行聚类分析 -

(本文所使用的Python库和版本号: Python 3.6, Numpy 1.14, scikit-learn 0.19, matplotlib 2.2)

前面的机器学习类文章（编号从010-019）都是关于**监督学习**，但是从本篇文章开始，开始讲解**无监督学习**方面，无监督学习是指处理的数据没有任何形式的标记，我们没有对训练数据集进行实现的类别划分，故而相当于抹黑处理，要让机器学习自己找出样本所属的类别，那么机器学习通过什么方式来找出“所属类别”了？这就是聚类算法的作用了。

介绍

聚类算法，其核心思想就是中国的“人以类聚，物以群分”，就是机器学习将通过一定的算法来将样本划分类别，使得相互之间**相似**的样本划分为一个类别，**不相似**的样本划分为不同的类别中。

K-means算法是最流行的聚类算法之一，这种算法常常利用数据的不同属性将输入数据划分为K组，这种划分是使用最优化的技术实现的，让各组内的数据点与该组中心点的距离平方和最小化。

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。
k-means算法特点在于：同一聚类的簇内的对象相似度较高；而不同聚类的簇内的对象相似度较小。

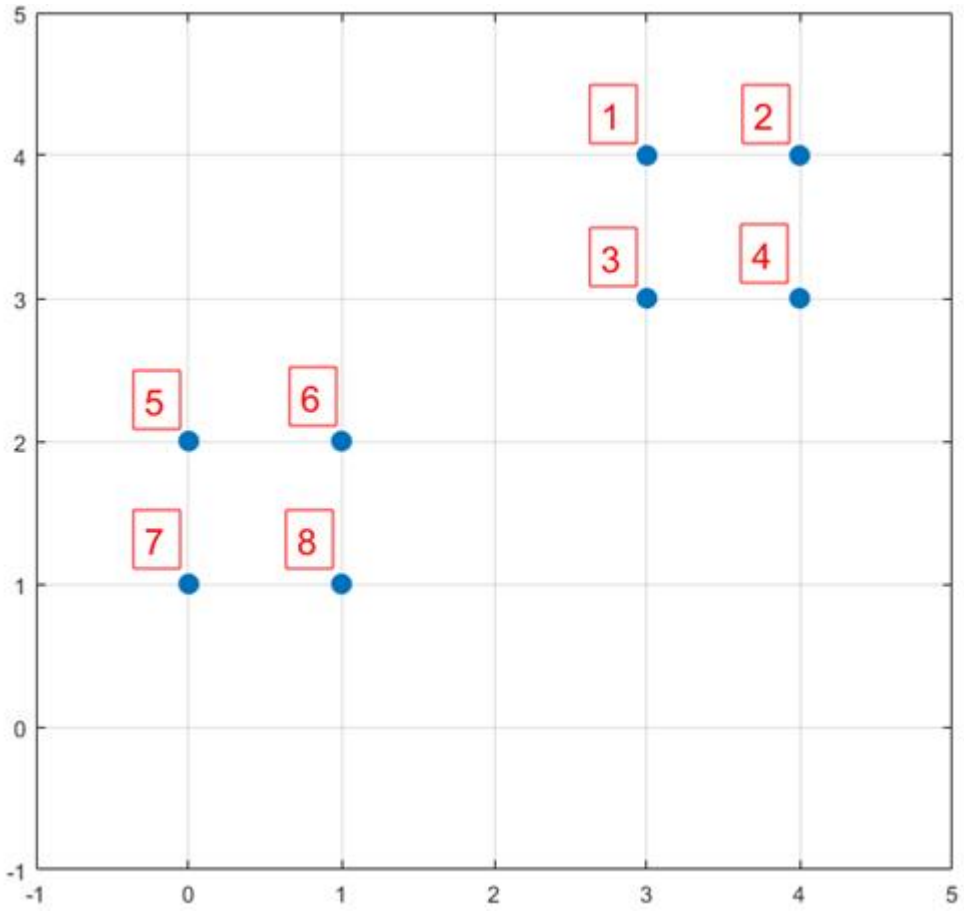
算法背景

原始的 K-means算法最开始随机选取数据集中K个点作为聚类中心;

K-means++算法描述如下：

- 步骤一：随机选取一个样本作为第一个聚类中心 c1；
- 步骤二：计算每个样本与当前已有类聚中心最短距离（即与最近一个聚类中心的距离），用 D(x)表示；这个值越大，表示被选取作为聚类中心的概率较大；最后，用轮盘法选出下一个聚类中心；
- 步骤三：重复步骤二，知道选出 k 个聚类中心。

案例分析：



假设6号点被选择为第一个初始聚类中心，那在进行步骤二时每个样本的***D(x)***和被选择为第二个聚类中心的概率如下表所示：

序号	①	②	③	④	⑤	⑥	⑦	⑧
$D(x)$	$2\sqrt{2}$	$\sqrt{13}$	$\sqrt{5}$	$\sqrt{10}$	1	0	$\sqrt{2}$	1
$D(x)^2$	8	13	5	10	1	0	2	1
$P(x)$	0.2	0.325	0.125	0.25	0.025	0	0.05	0.025
Sum	0.2	0.525	0.65	0.9	0.925	0.925	0.975	1

其中的**P(x)**就是每个样本被选为下一个聚类中心的概率。最后一行的**Sum**是概率**P(x)**的累加和，用于轮盘法选择出第二个聚类中心。（轮盘法是随机产生出一个0~1之间的随机数，判断它属于哪个区间，那么该区间对应的序号就是被选择出来的第二个聚类中心了。例如1号点的区间为[0,0.2)，2号点的区间为[0.2, 0.525)。）

从上表可以直观的看到第二个初始聚类中心是1号，2号，3号，4号中的一个的概率为0.9。而这4个点正好是离第一个初始聚类中心6号点较远的四个点。这也验证了K-means的改进思想：即离当前已有聚类中心较远的点有更大的概率被选为下一个聚类中心。

1. 准备数据集

本次所使用的数据集是我前面的文章**[机器学习010-用朴素贝叶斯分类器解决多分类问题]**中所采用的数据集，一个具有四种不同类别，两种不同features的小数据集，其加载方法和显示方法如下所示。

```
# 准备数据集
data_path='E:\PyProjects\DataSet\FireAI/data_multivar.txt'
df=pd.read_csv(data_path,header=None)
dataset_X,dataset_y=df.iloc[:,:-1],df.iloc[:,-1]
print(dataset_X.info())
print('-'*100)
print(dataset_y.head())
dataset_X=dataset_X.values
dataset_y=dataset_y.values
```

----------输--------出----------

RangeIndex: 400 entries, 0 to 399 Data columns (total 2 columns): 0 400 non-null float64 1 400 non-null float64 dtypes: float64(2) memory usage: 6.3 KB None

----------完----------

表明结果数据集已经正确地加载到内存中，且每一个features中都没有Null值，我们无需做进一步的缺失值处理。

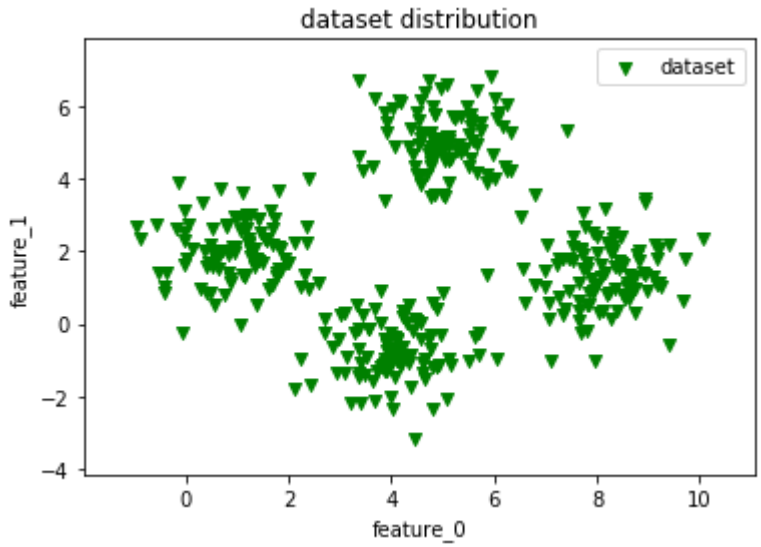
下面将这个数据集进行2D可视化，如下是可视化的代码：

```
# 无标签数据集可视化，将第一列feature作为X，第二列feature作为y
def visual_2D_dataset_dist(dataset):
    '''将二维数据集dataset显示在散点图中'''
    assert dataset.shape[1]==2,'only support dataset with 2 features'
    plt.figure()
    X=dataset[:,0]
    Y=dataset[:,1]
    plt.scatter(X,Y,marker='v',c='g',label='dataset')

    X_min,X_max=np.min(X)-1,np.max(X)+1
    Y_min,Y_max=np.min(Y)-1,np.max(Y)+1
    plt.title('dataset distribution')
    plt.xlim(X_min,X_max)
    plt.ylim(Y_min,Y_max)
    plt.xlabel('feature_0')
    plt.ylabel('feature_1')
    plt.legend()

visual_2D_dataset_dist(dataset_X)
```

得到的结果如下：



########################小********结################################**

1. 本数据集的加载很简单，只需用Pandas就可以直接加载，且不需要做其他处理。

2. 此处需要注意，无标签数据集的二维平面可视化，不能使用label数据，故而此处的可视化函数和我以往文章中的可视化函数是不一样的，此处需要额外注意。

3. 从二维平面散点图中可以看出，这个数据集大概可以分为4个不同的类别，即数据都分布在四个族群里，这就是我们可以用K-mean算法的基础。

###

2. 构建K-means算法

构建K-means算法的过程很简单，和其他的SVM，随机森林算法的构建方式一样，如下代码：

```
# 定义一个k-means对象
from sklearn.cluster import KMeans
#init='k-means++':初始聚类中心(尽可能远)，也是默认值
#init：有三个可选值：’k-means++’， ‘random’，或者传递一个ndarray向量。
# 此参数指定初始化方法，默认值为 ‘k-means++’
#n_clusters：整形，缺省值=8 （生成的聚类数，即产生的质心（centroids）数）
#n_init：整形，缺省值=10 ，用不同的质心初始化值运行算法的次数，选出最优结果。
kmeans=KMeans(init='k-means++',n_clusters=4,n_init=10)
# 这几个参数是初始化设定的，其中n_clusters是从二维散点图中看出大概有4个族群
kmeans.fit(dataset_X)
```

----------输--------出----------

KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300, n_clusters=4, n_init=10, n_jobs=1, precompute_distances='auto', random_state=None, tol=0.0001, verbose=0)

----------完----------

虽然此处我们定义了一个KMeans对象，且使用我们的无标签数据集进行了训练，可是训练结果怎么样了？我们怎么知道k-means算法是否正确的划分了不同类别？

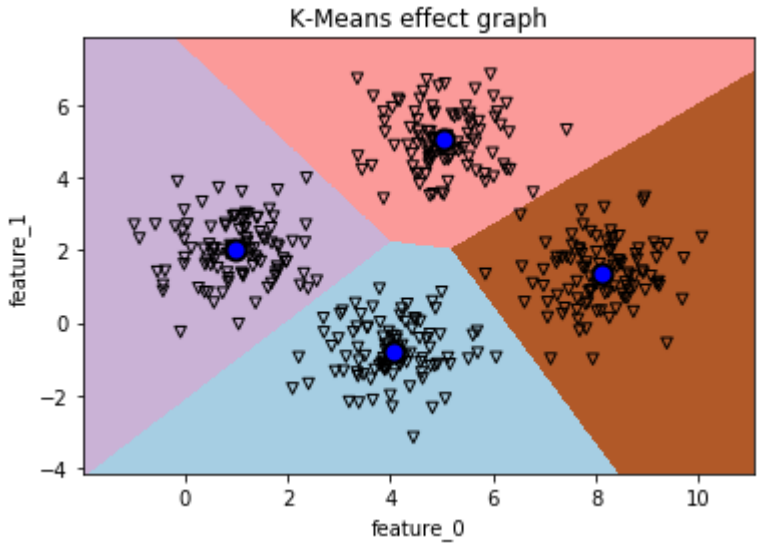
所以我们需要一个可视化的结果，就像前面文章中提到的SVM分类结果图一样，此处我们定义了一个专门用于可视化K-means聚类结果的函数，并用该函数来查看此处聚类的效果。代码如下：

```
def visual_kmeans_effect(k_means,dataset):
    assert dataset.shape[1]==2,'only support dataset with 2 features'
    X=dataset[:,0]
    Y=dataset[:,1]
    X_min,X_max=np.min(X)-1,np.max(X)+1
    Y_min,Y_max=np.min(Y)-1,np.max(Y)+1
    # meshgrid 生成网格点坐标矩阵
    X_values,Y_values=np.meshgrid(np.arange(X_min,X_max,0.01),
                                  np.arange(Y_min,Y_max,0.01))
    # 预测网格点的标记
    predict_labels=k_means.predict(np.c_[X_values.ravel(),Y_values.ravel()])
    predict_labels=predict_labels.reshape(X_values.shape)
    plt.figure()
    plt.imshow(predict_labels,interpolation='nearest',
               extent=(X_values.min(),X_values.max(),
                       Y_values.min(),Y_values.max()),
               cmap=plt.cm.Paired,
               aspect='auto',
               origin='lower')

    # 将数据集绘制到图表中
    plt.scatter(X,Y,marker='v',facecolors='none',edgecolors='k',s=30)

    # 将中心点绘制到图中
    centroids=k_means.cluster_centers_
    plt.scatter(centroids[:,0],centroids[:,1],marker='o',
                s=100,linewidths=2,color='k',zorder=5,facecolors='b')
    plt.title('K-Means effect graph')
    plt.xlim(X_min,X_max)
    plt.ylim(Y_min,Y_max)
    plt.xlabel('feature_0')
    plt.ylabel('feature_1')
    plt.show()

visual_kmeans_effect(kmeans,dataset_X)
```



########################小********结################################**

1. 定义K-means聚类算法的方法很简单，只需要从sklearn.cluster中导入KMeans，并定义一个KMeans对象即可，直接用fit()函数可以直接训练。

2. 此处使用k-means聚类算法对数据进行了聚类分析，可以使用函数visual_kmeans_effect()来直接查看聚类后的效果图。

3. 虽然可以直观看到效果图，但效果图还是难以量化k-means聚类算法的准确度，这些内容将在后续文章中讲解。

###

k-means的优点:

K-Means聚类算法的优点主要集中在:　1.算法快速、简单;　2.对大数据集有较高的效率并且是可伸缩性的;　3.时间复杂度近于线性，而且适合挖掘大规模数据集。K-Means聚类算法的时间复杂度是O(n×k×t) ,其中n代表数据集中对象的数量，t代表着算法迭代的次数，k代表着簇的数目

k-means的缺点:

1、在 K-means 算法中 K 是事先给定的，这个 K 值的选定是非常难以估计的。很多时候，事先并不知道给定的数据集应该分成多少个类别才最合适。　　2、 在 K-means 算法中，首先需要根据初始聚类中心来确定一个初始划分，然后对初始划分进行优化。这个初始聚类中心的选择对聚类结果有较大的影响，一旦初始值选择的不好，可能无法得到有效的聚类结果，这也成为 K-means算法的一个主要问题。
3、从 K-means 算法框架可以看出，该算法需要不断地进行样本分类调整，不断地计算调整后的新的聚类中心，因此当数据量非常大时，算法的时间开销是非常大的。所以需要对算法的时间复杂度进行分析、改进，提高算法应用范围。

参考资料:

1, Python机器学习经典实例，Prateek Joshi著，陶俊杰，陈小莉译