山东大学计算机科学与技术学院

数据挖掘实验报告

实验题目： Homework 3: Clustering with sklearn		学号： 201814852
日期：2018.12.14	班级： 18 级学硕班	姓名： 周培衍
Email：zhoupeiyan123@hotmail.com		

实验目的：

1） 测试 sklearn 中以下聚类算法在 tweets 数据集上的聚类效果。

2） 使用 NMI(Normalized Mutual Information)作为评价指标。

硬件环境：

Inter Core i7-7700 @ 3.60GHz 四核 + Dell 062KRH + 4G + P3-256

软件环境：

Windows 10 + Git + Anaconda + Spyder

实验步骤与内容：

一、 数据集

1）数据集 Tweets

数据集中是以 json 形式存储，记录的是已经处理好的文本以及类别。

The Tweets dataset is in format of JSON like follows:

--- {"text": "centrepoint winter white gala london", "cluster": 65}

--- {"text": "mourinho seek killer instinct", "cluster": 96}

--- {"text": "roundup golden globe won seduced voice", "cluster": 72}

--- {"text": "disruption storm cold air sweep south florida", "cluster": 140}

2）预处理

首先将 Tweets 中的内容读取出来，将文本保存到一个 list 中

text_list {″ centrepoint winter″, ″ seek killer″,...}

将原本分类保存到一个 list 中，方便以后评估

clu_list {65, 96,...}

因为后面的比较需要将文本作为向量处理，所以需要将 tfidf 矩阵和 count(词频)矩阵调用 sklearn 已有的方法(TfidfVectorizer, CountVectorizer)计算出来，其中的分类器采用之前实验采用的 jieba 工具。到这里数据的预处理工作就做完了。

二、 Cluster with sklearn

1) Sklearn

sklearn，全称 scikit-learn，是一款基于 python 的数据处理库，安装需要相应版本的 numpy、scipy、vc++，采用 pip install 下载好的 whl 文件，因为直接 pip 或者 easy-install 的话会很慢，而且版本也有可能不兼容。

这里会有相应的版本 whl 文件，是由 LFD 推出的很好的工具

https://www.lfd.uci.edu/~gohlke/pythonlibs

2) Clusters

安装 sklearn 之后，进入 scikit-learn 官方的学习模块

https://scikit-learn.org/stable/modules/clustering.html

官方文档很少有错，而且讲的很细，实在有不懂得 StackOverFlow 和 blog 也是很好的工具。

下面需要做的就是看文档和 demo,一共用的 7 个方法测试调参看效果。

1-K-means，

2-AffinityPropagation，

3-MeanShift，

4-SpectralClustering，

5-AgglomerativeClustering，

6-DBSCAN，

7-Birch

前面的数字为设置的 mode，区分不同的方法。

K-means: 把 n 个点（可以是样本的一次观察或一个实例）划分到 *k* 个聚类中，使得每个点都属于离他最近的均值（此即聚类中心）对应的聚类，以之作为聚类的标准。这个问题将归结为一个把数据空间划分为 Voronoi cells 的问题。需要调参 n_clusters 找到最好的效果。

--- n_clusters: 指定 K 的值

--- max_iter: 对于单次初始值计算的最大迭代次数

--- n_init: 重新选择初始值的次数

--- init: 制定初始值选择的算法

--- n_jobs: 进程个数，为-1 的时候是指默认跑满 CPU

AffinityPropagation: 与诸如 k 均值或 k-medoids 之类的聚类算法不同，亲和度传播不需要在运行算法之前确定或估计聚类的数量。 与 k-medoids 类似，亲和传播找到“范例”，输入集的成员代表集群。不需调参，默认设置就可以。

--- damping: 衰减系数，默认为 0.5

--- convergence_iter: 迭代次后聚类中心没有变化，算法结束，默认为 15

--- max_iter: 最大迭代次数，默认 200

--- copy: 是否在元数据上进行计算，默认 True，在复制后的数据上进行计算

--- preference: S 的对角线上的值
--- affinity: 相似度(S)矩阵，默认为 euclidean（欧氏距离）矩阵
MeanShift: 平均移位是用于定位密度函数的最大值的非参数特征空间分析技术，即所谓的模式搜索算法。应用领域包括计算机视觉和图像处理中的聚类分析。简单调参(quantile=0.2, n_samples=500)，不过 tfidf_matrix 会报 sparse 的 bug，所以就用词频矩阵(count_matrix)测试，效果也并不好，可能是太稀疏的原因。
--- bandwidth: float，高斯核函数的带宽，如果没有给定，则使用 sklearn.cluster.estimate_bandwidth 自动估计带宽
--- seeds: array, 我理解的 seeds 是初始化的质心，如果为 None 并且 bin_seeding=True,就用 clustering.get_bin_seeds 计算得到
--- bin_seeding: boolean，在没有设置 seeds 时起作用，如果 bin_seeding=True,就用 clustering.get_bin_seeds 计算得到质心，如果 bin_seeding=False，则设置所有点为质心
--- min_bin_freq: int，clustering.get_bin_seeds 的参数，设置的最少质心个数
SpectralClustering: 谱聚类技术利用数据的相似性矩阵的谱（特征值）来在较少维度的聚类之前执行维数降低。 提供相似性矩阵作为输入，并且包括对数据集中每对点的相对相似性的定量评估。当相似矩阵建立方式(affinity)为内置高斯核函数'rbf'时，调参 n_clusters；为 K 邻近法'nearest_neighbors' 时，还需调参 n_neighbors。
--- n_clusters: 代表我们在对谱聚类切图时降维到的维数，同时也是最后一步聚类算法聚类到的维数。也就是说 scikit-learn 中的谱聚类对这两个参数统一到了一起。简化了调参的参数个数。虽然这个值是可选的，但是一般还是推荐调参选择最优参数。
--- affinity: 也就是我们的相似矩阵的建立方式。可以选择的方式有三类，第一类是 'nearest_neighbors'即 K 邻近法。第二类是'precomputed'即自定义相似矩阵。第三类是全连接法，可以使用各种核函数来定义相似矩阵，还可以自定义核函数。最常用的是内置高斯核函数'rbf'。其他比较流行的核函数有‘linear’即线性核函数，‘poly’即多项式核函数，‘sigmoid’即 sigmoid 核函数。如果选择了这些核函数， 对应的核函数参数在后面有单独的参数需要调。affinity 默认是高斯核'rbf'。一般来说，相似矩阵推荐使用默认的高斯核函数。
--- gamma: 如果我们在 affinity 参数使用了多项式核函数 'poly'，高斯核函数‘rbf’，或者'sigmoid'核函数，那么我们就需要对这个参数进行调参。
--- n_neighbors: 如果我们 affinity 参数指定为'nearest_neighbors'即 K 邻近法，则我们可以通过这个参数指定 KNN 算法的 K 的个数。默认是 10.我们需要根据样本的分布对这个参数进行调参。如果我们 affinity 不使用'nearest_neighbors'，则无需理会这个参数。
--- n_init: 即使用 K-Means 时用不同的初始值组合跑 K-Means 聚类的次数，这个和 K-Means 类里面 n_init 的意义完全相同，默认是 10，

一般使用默认值就可以。如果你的 n_clusters 值较大，则可以适当增大这个值。

AgglomerativeClustering: Hierarchical clustering 层次聚类（也称为层次聚类分析或 HCA）是一种聚类分析方法，旨在构建聚类层次结构。Agglomerative 凝聚性：这是一种“自下而上”的方法：每个观察都在它自己的集群中开始，并且当一个集群向上移动时，它们将被合并。Sklearn 中只考虑了 Agglomerative 这一种，没考虑 Divisive。需要调参 linkage 和 n_clusters。tfidf_matrix 也会报 sparse 的 bug，所以就也用词频矩阵(count_matrix)测试

--- n_clusters: 一个整数，指定分类簇的数量

--- connectivity: 一个数组或者可调用对象或者 None，用于指定连接矩阵

--- affinity: 一个字符串或者可调用对象，用于计算距离。可以为: 'euclidean', 'l1', 'l2', 'mantattan', 'cosine', 'precomputed', 如果 linkage='ward'，则 affinity 必须为'euclidean'

--- linkage: 一个字符串，用于指定链接算法

'single': 单链接 single-linkage，采用 dmindmin

'complete': 全链接 complete-linkage 算法，采用 dmaxdmax

'average': 均连接 average-linkage 算法，采用 davg

'ward': 最小化被合并的 clusters 的方差

DBSCAN: 英文全写为 Density-based spatial clustering of applications with noise，这个算法是以密度为本的：给定某空间里的一个点集合，这算法能把附近的点分成一组（有很多相邻点的点），并标记出位于低密度区域的局外点（最接近它的点也十分远）。简单调参(eps=1, min_samples=1)。

--- eps: DBSCAN 算法参数，即我们的 ϵ-邻域的距离阈值，和样本距离超过 ϵ 的样本点不在 ϵ-邻域内。默认值是 0.5,一般需要通过在多组值里面选择一个合适的阈值。eps 过大，则更多的点会落在核心对象的 ϵ-邻域，此时我们的类别数可能会减少，本来不应该是一类的样本也会被划为一类。反之则类别数可能会增大，本来是一类的样本却被划分开。

--- min_samples: DBSCAN 算法参数，即样本点要成为核心对象所需要的 ϵ-邻域的样本数阈值。默认值是 5,一般需要通过在多组值里面选择一个合适的阈值。通常和 eps 一起调参。在 eps 一定的情况下，min_samples 过大，则核心对象会过少，此时簇内部分本来是一类的样本可能会被标为噪音点，类别数也会变多。反之 min_samples 过小，则会产生大量的核心对象，导致类别数过少。

Birch: 英文全称：balanced iterative reducing and clustering using hierarchies，中文：利用层次方法的平衡迭代规约和聚类。是一个非监督式分层聚类算法，算法的优势在于能够利用有限的内存资源完成对大数据集的高质量的聚类。该算法通过构建聚类特征树（Clustering Feature Tree，简称 CF Tree），在接下来的聚类过程中，直接对聚类特征进行聚类，无需对原始数据集进行聚类。因此在多数情况下只需扫描一次数据库即可进行聚类，IO 成本与数据集尺寸呈线性关系。

简单调参(n_clusters=90, threshold=0.7)。

--- threshold：即叶节点每个 CF 的最大样本半径阈值 T，它决定了每个 CF 里所有样本形成的超球体的半径阈值。一般来说 threshold 越小，则 CF Tree 的建立阶段的规模会越大，即 BIRCH 算法第一阶段所花的时间和内存会越多。但是选择多大以达到聚类效果则需要通过调参决定。默认值是 0.5.如果样本的方差较大，则一般需要增大这个默认值。

--- branching_factor：即 CF Tree 内部节点的最大 CF 数 B，以及叶子节点的最大 CF 数 L。这里 scikit-learn 对这两个参数进行了统一取值。也就是说，branching_factor 决定了 CF Tree 里所有节点的最大 CF 数。默认是 50。如果样本量非常大，比如大于 10 万，则一般需要增大这个默认值。选择多大的 branching_factor 以达到聚类效果则需要通过和 threshold 一起调参决定

--- n_clusters：即类别数 K，在 BIRCH 算法是可选的，如果类别数非常多，我们也没有先验知识，则一般输入 None，此时 BIRCH 算法第 4 阶段不会运行。但是如果我们有类别的先验知识，则推荐输入这个可选的类别值。默认是 3，即最终聚为 3 类。

3) cluster 比较及评估方法

通过运行，得到一下效果

```
In [1]: runfile('D:/projects/python/repository/201814852ZhouPeiyan/Homework3/__init__.py',
wdir='D:/projects/python/repository/201814852ZhouPeiyan/Homework3')
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\dell\AppData\Local\Temp\jieba.cache
共有2472条数据
其中已知分类有89种
Loading model cost 0.607 seconds.
Prefix dict has been built successfully.
选择mode为1(K-means)时，采用NMI评测方法，best_k为84时正确率最高，为0.7796151899630053
选择mode为2(AffinityPropagation)时，采用NMI评测方法，正确率为0.7624742003503602
选择mode为3(MeanShift)时，采用NMI评测方法，正确率为0.10152371785802763
选择mode为4(SpectralClustering)时，affinity为高斯核函数'rbf'
采用NMI评测方法，best_k为83时正确率最高为0.6948181961967183
选择mode为4(SpectralClustering)时，affinity为K邻近法'nearest_neighbors'
采用NMI评测方法，best_k为59，besk_n为6时正确率最高为0.8511768223931282
选择mode为5(AgglomerativeClustering)时，linkage为ward
采用NMI评测方法，best_k为86时正确率最高，为0.5699325276708949
选择mode为6(DBSCAN)时，采用NMI评测方法，正确率为0.7601013610704973
选择mode为7(Birch)时，采用NMI评测方法，best_k为90时正确率最高，为0.7685330121788387
```

其中通过 NMI 评估分类最好的是 **SpectralClustering** 在 affinity 为 K 近邻法，n_clusters 为 59，n_neighbors 为 6 时，NMI 达到 0.851。效果最差的 **MeanShift**，NMI 为 0.102，可能是稀疏的问题，或者别的地方出问题了，但太差了。次差的是 **AgglomerativeClustering**，NMI 为 0.570，但这个方法相比较其他的运行时间太长了。比这个稍微强一点的是 **SpectralClustering** 在 affinity 为高斯核函数，NMI 最好为 0.694。剩下的都还算挺好，差别也不大，分别为 **K-means**、**Birch**、**AffinityPropagation**、**DBSCAN**，NMI 分别为 0.779、0.768、0.762、0.760。

结论分析与体会：
Homework3 采用了 sklearn 库，相关 cluster 方法不用自己实现了，大大的减轻了代码量，也能明白这些方法的原理和优缺点以及适用 cluster 的数据类型，不过也有点不好的地方，一是代码量减少不少，二是有一些评估方法(如欧式距离)，不能采取别的可能更适合的方法(如余弦值)，sklearn 并没有实现。总体来说还是很有收获，不过对于 MeanShift 出现的问题还是需要在问问同学和老师(也有别的同学出现类似的情况，NMI 很小)。