clustering

1.实验目的：

学习并使用 python 中的 sklearn 包，测试 sklearn 中以下聚类算法在 tweets 数据集上的聚类效果，使用 NMI(Normalized Mutual Information)作为评价指标。学习并使用 python 中的 sklearn 包，

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers

2.实现过程：

1）处理数据集

通过 get_data():函数读取数据，把数据集中的待聚类的文本信息存储在 feature 列表中，把每一个文本信息所属的簇存储在 cluster 列表中。然后调用 tfidf_vectorizer.fit_transform(feature)函数把文本信息转换成 tfidf 稀疏矩阵，然后通过 tfidf_matrix.toarray() 函数将稀疏矩阵形式转换为二维数组形式。最后函数返回文本信息的 tfidf 的矩阵 tfidf_matrix 和存储文本信息所在的簇的 cluster 列表，具体代码实现如下：

```
def get_data():

    final = open('Tweets.txt','r')

    data = [line.strip().split('\t') for line in final]
    feature = []
    cluster = []
    for i in range(len(data)):
        temp = data[i][0]
        temp = temp.split(':')
        feature.append(temp[1][2:temp[1].index(',') - 1])
        cluster.append(int(temp[2][1:temp[2].index('}')]))
    #tokenizer: 指定分词函数; lowercase: 在分词之前将所有的文本转换成小写
    tfidf_vectorizer = TfidfVectorizer(tokenizer=word_tokenize,
                                       stop_words='english',lowercase=True)

    tfidf_matrix = tfidf_vectorizer.fit_transform(feature)
    tfidf_matrix = tfidf_matrix.toarray()
```

2）各种聚类方法的实现：

1.kmeans：

实现代码：

```
def kmeans(tfidf_matrix):
    num_clusters = 89 #聚类个数
    km_cluster = KMeans(n_clusters=num_clusters, max_iter=300, n_init=10,
                        init='k-means++',)
    return km_cluster.fit_predict(tfidf_matrix)
```

其中返回的是通过 kmeans 聚类方法得到的各个文本的标签，然后

```
print('kmeans聚类算法的成功率: ',end = "")
print(metrics.normalized_mutual_info_score(cluster, result_km ))
```

使用 NMI 作为聚类效果的评价标准，调用

metrics.normalized_mutual_info_score(cluster, result_hc)函数。

其中各个参数的含义：

1) n_clusters: 即我们的 k 值，一般需要多试一些值以获得较好的聚类效果。在此选择的 k 值为 89.

2）max_iter： 最大的迭代次数，一般如果是凸数据集的话可以不管这个值，如果数据集不是凸的，可能很难收敛。在此设最大的迭代

次数为 300 次。

3）n_init：用不同的初始化质心运行算法的次数。由于 K-Means 是结果受初始值影响的局部最优的迭代算法，因此需要多跑几次以选择一个较好的聚类效果，默认是 10，在此使用默认值。

4）init： 即初始值选择的方式，可以为完全随机选择'random',优化过的'k-means++'或者自己指定初始化的 k 个质心。使用默认的'k-means++'。

5）algorithm：有“auto”，“full” or “elkan”三种选择。直接采用“auto”即可。

2. affinityaropagation：

实现代码：

```
def affinityaropagation(tfidf_matrix):
    ap_cluster = AffinityPropagation(damping=0.5, max_iter=200, convergence_iter=15,
        copy=True, preference=None, affinity='euclidean', verbose=False)
    print('affinityaropagation聚类的个数: ',end = "")
    print(len(set(ap_cluster.fit_predict(tfidf_matrix)))) #聚类的个数
    return ap_cluster.fit_predict(tfidf_matrix)
```

affinityaropagation 的聚类方法是不提前给出聚类个数的，所以在此函数中我们打印出聚类的个数。

其中各个参数的含义：

damping :阻尼系数,默认值 0.5，使用默认值。

max_iter :,最大迭代次数,默认值是 200。

copy :默认为 true,即允许对输入数据的复制。

affinity : string, optional, default=``euclidean``目前支持计算预欧几里得距离。 即点之间的负平方欧氏距离。

3.dnscan：

实现代码：

```
def dbscan(tfidf_matrix):

    ds_cluster = DBSCAN(eps = 0.99, min_samples = 1 )
    print('dbscan聚类的个数: ',end = "")
    print(len(set(ds_cluster.fit_predict(tfidf_matrix))))
    return ds_cluster.fit_predict(tfidf_matrix)
```

dbscan 的聚类方法是不提前给出聚类个数的，所以在此函数中我们打印出聚类的个数。

其中各个参数的含义：

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.

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.

在此如果我们使用默认参数，得到结果如下：

```
dbscan聚类的个数: 6
dbscan聚类算法的成功率: 0.10801213485085728
```

得到的聚类的个数太少，且准确率很低，需要对 DBSCAN 的两个关键的参数 eps 和 min_samples 进行调参。调参增加类别，有两个方向都是可以的，一个是继续减少 eps，另一个是增加 min_samples。当 min_samples 大于等于 2 是，聚类准确率很差，成功率不到百分之 50。

最后使用参数 eps = 0.99，min_samples = 1.

4. meanshift:

代码实现：

```
def meanshift(tfidf_matrix):
    ms_cluster = MeanShift(bandwidth=0.8, bin_seeding=True)
    print('meanshift聚类的个数: ',end = "")
    print(len(set(ms_cluster.fit_predict(tfidf_matrix))))
    return ms_cluster.fit_predict(tfidf_matrix)
```

其中各个参数含义：

bandwidth：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，在此给定参数 bandwidth = 0.8.

bin_seeding ：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

5. spectralclustering

代码实现：

```
def spectralclustering(tfidf_matrix):
    num_clusters = 89 #聚类个数
    sc_cluster = SpectralClustering(n_clusters=num_clusters,
                                    assign_labels="discretize",random_state=0)

    return sc_cluster.fit_predict(tfidf_matrix)
```

其中各个参数含义：

n_clusters:切图时降到的维数。在此 n_clusters = 89

assign_labels:最后使用的聚类方式。

6. hierarchicalclustering 和 agglomerativeclustering

代码实现：

```
def hierarchicalclustering(tfidf_matrix):
    num_clusters = 89 #聚类个数
    hc_cluster = AgglomerativeClustering(n_clusters=num_clusters, linkage='ward')
    return hc_cluster.fit_predict(tfidf_matrix)
```

```
def agglomerativeclustering(tfidf_matrix):
    num_clusters = 89
    ac_cluster = AgglomerativeClustering(n_clusters=num_clusters,linkage = 'average')
    return ac_cluster.fit_predict(tfidf_matrix)
```

hierarchical clustering 可在不同层次上对数据集进行划分，形成树状的聚类结构，AggregativeClustering 是一种常用的层次聚类算法。

linkage：一个字符串，用于指定链接算法

ward：单链接 single-linkage

complete：全链接 complete-linkage 算法

average：均连接 average-linkage 算法

其余参数采用默认参数，聚类效果就能达到一个不错的效果。

7. gaussianmixture

代码实现：

```
def gaussianmixture(tfidf_matrix):
    num_clusters = 89
    gm_cluster = GaussianMixture(n_components = num_clusters, covariance_type='diag')
    return gm_cluster.fit(tfidf_matrix).predict(tfidf_matrix)
```

其中各个参数含义：

n_components: 混合高斯模型个数，默认为 1，在此取值 89.

covariance_type: 协方差类型，包括 {‘full’，‘tied’，‘diag’，‘spherical’} 四种，在此 covariance_type 的类型为 diag，diag 指每个分量有各自不同对角协方差矩阵（非对角为零，对角不为零）。

3 实验结果

```
affinityaropagation聚类的个数：331
meanshift聚类的个数：612
dbscan聚类的个数：1461
kmeans聚类算法的成功率：0.7651880181395159
affinityaropagation聚类算法的成功率：0.7836988975391974
meanshift聚类算法的成功率：0.7013762593042232
hierarchicalclustering聚类算法的成功率：0.7839677498797781
spectralclustering聚类算法的成功率：0.7838965861576753
agglomerativeclustering聚类算法的成功率：0.8949194137166658
dbscan聚类算法的成功率：0.7538946164849132
gaussianmixture聚类算法的成功率：0.7804124808125011
```

其中没有给定聚类参数的算法 affinityaropagation 聚类个数为 331，meanshift 聚类个数为 612，dbscan 聚类个数为 1461，看似与给定的 89 个类相差较多，但使用 NMI 评价方法时聚类效果还可以，分别为 0.783,0.701,0.753。

各类聚类算法的成功率一般稳定在 0.7-0.8，其中 agglomerativeclustering 聚类方法效果最好，达到了 0.89。

4.总结

通过此次作业，比较系统的学习了 sklearn 包，对 sklean 包有了大概的了解以及基本的使用。在此次的学习中也遇到了一些困难，最后也通过看 sklearn 官网以及博客等方法解决了问题。sklearn 包也是以后在数据挖掘和机器学习的课程学习中很实用的一个工具，了解和学习 sklearn 包对我帮助很大，当然，在学习过程中也不能一直通过调用包来实现，更多算法也需要自己学习和用代码实现。