山东大学
SHANDONG UNIVERSITY

实验报告

课　　程：	Data Mining
实验题目：	Homework 3
姓　　名：	陈昕
学　　号：	201814806
班　　级：	2018 级计算机学硕班
实验时间：	2018/11/25 – 2018/12/24

一、 实验目的

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

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

通过该实验，掌握常用聚类算法的思想及调用方式。

二、 实验环境

硬件环境：

intel i5-7400 CPU @ 3.00 GHz
8.00 GB RAM

软件环境：

Windows 10 64 位 家庭版
python 3.6.6
Anaconda custom (64-bit)
Jupyter notebook 5.7.0
numpy 1.15.4
scikit-learn 0.20.0

三、 实验过程

准备工作

首先，读取 Tweets.txt，其中每一行是是一个 json，可以用 json.loads 读取。其中，text 指向文本，也就是我们要用到的数据，而 cluster 表示每条数据属于哪个类。

In [4]:
```
# construct tf-idf matrix
corpus = [d['text'] for d in data]
vectorizer = TfidfVectorizer(min_df=2, sublinear_tf=True)
X = vectorizer.fit_transform(corpus).todense()
tfidf_matrix = np.array(X)
```

因为文本不能直接用于聚类，所以我使用了 sklearn 中的 TfidfVectorizer 将文本数据转化为包含 tfidf 值的向量。使用 min_df=2 来保证只有文档频率大于 1 的词才会被加入字典，加快计算，sublinear_tf=True 来让 tf 被标准化。

因为这样生成的矩阵是稀疏的(scipy sparse matrix)，所以要用 todense()转化它，以便聚类方法的计算。最后用 np.array 来把矩阵转为 array-like 的。

In [5]:
```
# get labels of each tweet
labels = [d['cluster'] for d in data]
```

labels 是每条数据的实际标签，用于和聚类算法预测出的标签对比。

sklearn.metrics. **normalized_mutual_info_score** (*labels_true, labels_pred, average_method='warn'*) [source]

NMI(Normalized Mutual Information, 规范互信息)可以用于比较两个聚类结果的相似性，范围为 0.0（无互信息）-1.0（完美匹配）。我使用的 average_method 是‘arithmetic’，因为使用‘warn’会出现警告，而文档中介绍 0.22 版中将使用‘arithmetic’作为默认选项。

这种方法的主要缺点就是要知道数据真正属于哪个类，而很多场景下聚类就是因为不知道类信息。但是这一点对我们的数据来说刚刚好。

K-Means

In [6]:
```
# K-Means clustering test
k_means = KMeans(n_clusters=max(labels), n_jobs=-1).fit(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, k_means.labels_, average_method='arithmetic')
print('The NMI score of K-Means is:', nmi)
```

```
The NMI score of K-Means is: 0.8083287921393979
```

K-Means 聚类方法是很简单又高效的聚类方法，目标是最小化各点到类中心的距离的平方和。

但 sklearn 的实现没有办法指定使用的距离函数，所以只能用欧氏距离计算。尽管如此，最后算出的 NMI 分数为 0.8083287921393979，效果还是不错的。n_clusters 参数指定了要聚成几类，相当于 K；n_jobs 指定要使用几个核计算，-1 表示使用所有可用核。

Affinity propagation

In [7]:
```
# Affinity propagation clustering test
ap = AffinityPropagation().fit(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, ap.labels_, average_method='arithmetic')
print('The NMI score of Affinity propagation is:', nmi)
```

```
The NMI score of Affinity propagation is: 0.7733356331577484
```

Affinity propagation 通过在样本对之间发送数据创建集群，直到收敛。我没有修改默认参数，最后算出的 NMI 分数为 0.7733356331577484，效果

还行。

Mean Shift

```
In [8]: # MeanShift clustering test
        ms = MeanShift(bandwidth=0.44, bin_seeding=True, min_bin_freq=2, cluster_all=True, n_jobs=-1).fit(tfidf_matrix)
        nmi = normalized_mutual_info_score(labels, ms.labels_, average_method='arithmetic')
        print('The NMI score of MeanShift is:', nmi)
        print('Its shape is:', np.unique(ms.labels_).shape)

        The NMI score of MeanShift is: 0.6901183198703617
        Its shape is: (85,)
```

Mean Shift 聚类旨在发现样本的平滑密度中的“斑点”。它和 KMeans 一样是基于中心的，先更新中心候选为每个给定区域的点的平均值，然后在后处理阶段过滤候选来消除近似重复，形成最后的质心集。

为了选出质心，最重要的参数是 bandwidth，如果使用默认 bandwidth 或 bandwidth=1 都会导致最后结果只包含 1 个类。bandwidth=0.44 算是一个相对较好的 bandwidth，但 NMI 分数也只有 0.6901183198703617，属于较低的。

bin_seeding=True， min_bin_freq=2， n_jobs=-1 的设置是为了加速计算，而 cluster_all=True 的设置是为了让所有点都属于一个类。

Spectral Clustering

```
In [9]: # SpectralClustering test
        sc = SpectralClustering(n_clusters=max(labels)).fit(tfidf_matrix)
        nmi = normalized_mutual_info_score(labels, sc.labels_, average_method='arithmetic')
        print('The NMI score of SpectralClustering is:', nmi)

        The NMI score of SpectralClustering is: 0.698345651992622
```

谱聚类方法在样本之间进行低维度嵌入亲和度矩阵，然后在低维空间中进行 KMeans。根据文档描述，当各聚类结构高度非凸时，谱聚类十分有用。但对我们的数据，最后算出的 NMI 分数为 0.698345651992622，效果也不太好。

n_clusters 参数指定了要聚成几类。

Ward hierarchical clustering

```
In [10]: # Ward hierarchical clustering test
         whc = AgglomerativeClustering(n_clusters=max(labels), linkage='ward').fit(tfidf_matrix)
         nmi = normalized_mutual_info_score(labels, whc.labels_, average_method='arithmetic')
         print('The NMI score of Ward hierarchical clustering is:', nmi)

         The NMI score of Ward hierarchical clustering is: 0.7864984641747731
```

Ward hierarchical clustering 是层次聚类的一种，Ward 表示最小化所有聚类内的平方差的总和。这个优化目标与 K-Means 的目标很相似，但是计算方法是 agglomerative hierarchical approach。

最后算出的 NMI 分数为 0.7864984641747731，还算不错。

n_clusters 参数指定了要聚成几类，linkage='ward' 指定了链接方法。文档中提到了当 linkage 为 ward 时，只能使用 Euclidean 方式计算亲密性。

Agglomerative clustering

In [11]:
```
# AgglomerativeClustering test
ac = AgglomerativeClustering(n_clusters=max(labels), affinity='cosine', linkage='single').fit(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, ac.labels_, average_method='arithmetic')
print('The NMI score of Agglomerative Clustering with single linkage is:', nmi)
ac = AgglomerativeClustering(n_clusters=max(labels), affinity='cosine', linkage='average').fit(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, ac.labels_, average_method='arithmetic')
print('The NMI score of Agglomerative Clustering with average linkage is:', nmi)
```

```
The NMI score of Agglomerative Clustering with single linkage is: 0.22496764238754718
The NMI score of Agglomerative Clustering with average linkage is: 0.9003064131754916
```

凝聚聚类是一种通用的聚类算法系列，它通过连续合并或拆分嵌套聚类来构建嵌套聚类，上课时也重点讲了。这部分实验中我使用了其他 linkage 方式。

n_clusters 参数指定了要聚成几类，affinity='cosine'指定了用 cos 相似度来计算亲密性，linkage 选用了 single（最小化两类间最近点对的距离）和 average（最小化两类间所有点对的平均距离）方式。

可以看到，使用 single 方式计算出的 NMI 分数很低，大约为 0.23；而使用 average 方式计算出的分数就很高，大约为 0.9，这也是我计算出的最高 NMI 分数。

DBSCAN

In [12]:
```
# DBSCAN test
dbscan = DBSCAN(eps=0.1, n_jobs=-1).fit(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, dbscan.labels_, average_method='arithmetic')
print('The NMI score of DBSCAN with eps=0.1 is:', nmi)
print('Its shape is:', np.unique(dbscan.labels_).shape)
dbscan = DBSCAN(eps=1, n_jobs=-1).fit(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, dbscan.labels_, average_method='arithmetic')
print('The NMI score of DBSCAN with eps=1 is:', nmi)
print('Its shape is:', np.unique(dbscan.labels_).shape)
dbscan = DBSCAN(eps=2, n_jobs=-1).fit(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, dbscan.labels_, average_method='arithmetic')
print('The NMI score of DBSCAN with eps=2 is:', nmi)
print('Its shape is:', np.unique(dbscan.labels_).shape)
```

```
The NMI score of DBSCAN with eps=0.1 is: 0.027537116536474605
Its shape is: (5,)
The NMI score of DBSCAN with eps=1 is: 0.568925646557483
Its shape is: (64,)
The NMI score of DBSCAN with eps=2 is: -8.557811214631876e-17
Its shape is: (1,)
```

DBSCAN 算法将 cluster 视为由低密度区域分隔的高密度区域，高密度区域的连接半径由 eps 定义。

可以看出，当 eps=0.1 时和 eps=2 时，分类效果都非常差，NMI 很低。

而 eps=1 时，NMI 分数相对较高，为 0.568925646557483，但相对于其他算法来说 DBSCAN 的聚类效果还是属于最差的一档。

Birch

In [13]:
```
# Birch test
brc = Birch(n_clusters=max(labels)).fit(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, brc.labels_, average_method='arithmetic')
print('The NMI score of Birch is:', nmi)
```

```
The NMI score of Birch is: 0.8050399281197088
```

Birch 为给定数据构建一个称为特征树（CFT， Characteristic Feature Tree）的树。数据基本上被有损压缩为一组 CF 节点。CF 节点有许多称为特征子集（CF Subclusters）的子集群，位于非终端 CF 节点中的这些 CF 子集可以将 CF 节点作为子节点。

这种方法最后算出的 NMI 分数为 0.8050399281197088，效果比较好。

n_clusters 参数指定了要聚成几类。

Gaussian Mixture

In [14]:
```
# Gaussian mixtures test
gm_labels = GaussianMixture(n_components=max(labels), covariance_type='tied').fit_predict(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, gm_labels, average_method='arithmetic')
print('The NMI score of Gaussian mixtures is:', nmi)
```

```
The NMI score of Gaussian mixtures is: 0.7987454617917258
```

Gaussian Mixture 是一个概率模型，实际上不属于 cluster 而是属于 mixture 包。它假设所有数据点是从具有未知参数的有限数量的高斯分布的混合生成的，也就是每个点有一定概率属于 a 类的同时属于 b 类。

这种方法最后算出的 NMI 分数为 0.7987454617917258，也属于效果较好的。

n_components 参数指定了里面包含几个 component，每个 component 为一个高斯核，相当于聚类算法中的 n_clusters。

当不指定 covariance_type 是，默认是对每个 component 都需要计算协方差矩阵，这会造成内存不足的错误。

```
# Gaussian mixtures test
gm_labels = GaussianMixture(n_components=max(labels)).fit_predict(tfidf_matrix)
nmi = normalized_mutual_info_score(labels, gm_labels, average_method='arithmetic')
print('The NMI score of Gaussian mixtures is:', nmi)
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-77-37d39106b86d> in <module>
      1 # Gaussian mixtures test
----> 2 gm_labels = GaussianMixture(n_components=max(labels)).fit_predict(tfidf_matrix)
      3 nmi = normalized_mutual_info_score(labels, ac.labels_, average_method='arithmetic')
      4 print('The NMI score of Gaussian mixtures is:', nmi)

~\Anaconda3\lib\site-packages\sklearn\mixture\base.py in fit_predict(self, X, y)
    243
    244                 log_prob_norm, log_resp = self._e_step(X)
--> 245                 self._m_step(X, log_resp)
    246                 lower_bound = self._compute_lower_bound(
    247                     log_resp, log_prob_norm)

~\Anaconda3\lib\site-packages\sklearn\mixture\gaussian_mixture.py in _m_step(self, X, log_resp)
    673         self.weights_, self.means_, self.covariances_ = (
    674             _estimate_gaussian_parameters(X, np.exp(log_resp), self.reg_covar,
--> 675                                           self.covariance_type))
    676         self.weights_ /= n_samples
    677         self.precisions_cholesky_ = _compute_precision_cholesky(

~\Anaconda3\lib\site-packages\sklearn\mixture\gaussian_mixture.py in _estimate_gaussian_parameters(X, resp, reg_covar, covariance_type)
    283                    "diag": _estimate_gaussian_covariances_diag,
    284                    "spherical": _estimate_gaussian_covariances_spherical
--> 285                    }[covariance_type](resp, X, nk, means, reg_covar)
    286     return nk, means, covariances
    287

~\Anaconda3\lib\site-packages\sklearn\mixture\gaussian_mixture.py in _estimate_gaussian_covariances_full(resp, X, nk, means, reg_covar)
    162     """
    163     n_components, n_features = means.shape
--> 164     covariances = np.empty((n_components, n_features, n_features))
    165     for k in range(n_components):
    166         diff = X - means[k]

MemoryError:
```

而 covariance_type='tied'让所有 component 共享协方差矩阵，这样就不会出现 Memory Error 了。

四、 实验结论

在本次实验中，我实验了 sklearn 中的多种聚类算法，并了解了 NMI 这一评价聚类算法效果的指标。

在我使用的这些方法中，效果最好的还是使用 linkage=‘average’，affinity=‘cosine’的凝聚聚类算法，最差的则是使用 linkage=‘single’的凝聚聚类算法和 DBSCAN。如果参数调节不好，那么 MeanShift 和 DBSCAN 都会产生很差的结果。

简单来说，在知道类标签数量的情况下，KMeans、Birch、Gaussian Mixture 和使用 linkage=‘average’的凝聚聚类算法可以达到很好的效果。如果不知道类标签数量，Affinity propagation 算法的效果相对比较好。

这次实验给我的感觉就是现在发现的聚类算法大多都需要控制参数，典型的比如 K-Means 的 k，而这些控制参数的调整其实不是很直观，如果我是有数据而不加以分析，是很难选出好的聚类算法，达到很好的聚类效果的。

所以，还是要从数据的特征入手，分析数据与聚类算法的相性。比如这次实验中使用的知道真实类标签的文本数据，就最适合有 cos 相似度计算的凝聚聚类方法，而且聚类间的链接方式使用 average 就比使用 single 这种找最小值的方式更好。而 DBSCAN 是一个效果不好的典型例子，它的假设是 cluster 是由低密度区域分隔的高密度区域，从直觉上来讲，文本数据在高维空间里很难是高密度的，所以无论怎么调节参数都不能获得很好的聚类效果。

通过该实验，我了解到了很多常用聚类算法的思想及调用方式，对聚类问题有了更深刻的认识，收获很大。