Dating Mining

Experiment 3 Clustering with scikit-learn

姓名：包琛

学号：201814800

班级：2018 级计算机学硕班

指导老师：尹建华

时间：2018 年 12 月 30 日

1 实验要求

测试 sklearn 中以下聚类算法在 tweets 数据集上的聚类效果。
使用 NMI 作为评价指标。

Homework3: Clustering with sklearn

- 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。

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

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

2 实验背景

2.1 sklearn

Scikit-learn 项目最早由数据科学家 David Cournapeau 在 2007 年发起，需要 NumPy 和 SciPy 等其他包的支持，是 Python 语言中专门针对机器学习应用而发展起来的一款开源框架。
Scikit-learn 的基本功能主要被分为六大部分：分类，回归，聚类，数据降维，模型选择和数据预处理。
其中，聚类是指自动识别具有相似属性的给定对象，并将其分组为集合，属于无监督学习的范畴，最常见的应用场景包括顾客细分和试验结果分组。目前 Scikit-learn 已经实现的算法包括：K-均值聚类，谱聚类，均值偏移，分层聚类，DBSCAN 聚类等。

2.2 Normalized Mutual Information

Normalized Mutual Information(NMI)标准化互信息，常用在聚类中，度量 2 个聚类结果的相近程度。输入 2 个向量，每个向量的第 i 位表示第 i 个点归属的类。
首先要计算 Mutual Information (MI) 互信息，用来衡量两个数据分布的吻合程度。假设 U 与 V 是对 N 个样本标签的分配情况，则两种分布的熵（熵表示的是不确定程度）分别为：

$$H(U) = \sum_{i=1}^{|U|} P(i) \log(P(i))$$

$$H(V) = \sum_{j=1}^{|V|} P'(j) \log(P'(j))$$

其中，

$$P(i) = |U_i|/N$$

$$P'(j) = |V_j|/N$$

U 与 V 之间的互信息（MI）定义为：

$$\mathrm{MI}(U,V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} P(i,j) \log\left(\frac{P(i,j)}{P(i)P'(j)}\right)$$

其中，

$$P(i,j) = |U_i \cap V_j|/N$$

标准化后的互信息（Normalized mutual information）为：

$$\mathrm{NMI}(U,V) = \frac{\mathrm{MI}(U,V)}{\sqrt{H(U)H(V)}}$$

参考文献：

1. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html
2. https://www.jianshu.com/p/b9528df2f57a

1. k-means

```
sklearn.cluster.KMeans(n_clusters=8,
        init='k-means++',
       n_init=10,
       max_iter=300,
       tol=0.0001,
       precompute_distances='auto',
       verbose=0,
       random_state=None,
       copy_x=True,
       n_jobs=1,
       algorithm='auto'
       )
```

参数：

- n_clusters:簇的个数；
- init：初始簇中心的获取方法；
- n_init：获取初始簇中心的更迭次数，为了弥补初始质心的影响，算法默认会初始 10 次质心，实现算法，然后返回最好的结果；
- max_iter：最大迭代次数（因为 kmeans 算法的实现需要迭代）；
- tol：容忍度，即 kmeans 运行准则收敛的条件；
- precompute_distances：是否需要提前计算距离，这个参数会在空间和时间之间做权衡，如果是 True 会把整个距离矩阵都放到内存中，auto 会默认在数据样本大于 featurs*samples 的数量大于 12e6 的时候 False，False 时核心实现的方法是利用 Cpython 来实现的；
- verbose：冗长模式；
- random_state：随机生成簇中心的状态条件；
- copy_x：对是否修改数据的一个标记，如果 True，即复制了就不会修改数据。bool 在 scikit-learn 很多接口中都会有这个参数的，就是是否对输入数据继续 copy 操作，以便不修改用户的输入数据；
- n_jobs：并行设置；
- algorithm：kmeans 的实现算法，有：'auto'，'full'，'elkan'，其中 'full'表示用 EM 方式实现；

参考文献：https://blog.csdn.net/sinat_26917383/article/details/70240628

2. Affinity propagation

sklearn.cluster.AffinityPropagation(damping=0.5,
max_iter=200,
convergence_iter=15,
copy=True,
preference=None,
affinity='euclidean',
verbose=False)

参数:

- damping : float, optional, default: 0.5,阻尼系数,默认值 0.5
- max_iter : int, optional, default: 200,最大迭代次数,默认值是 200
- convergence_iter : int, optional, default: 15,在停止收敛的估计集群数量上没有变化的迭代次数。默认 15
- copy : boolean, optional, default: True,布尔值,可选,默认为 true,即允许对输入数据的复制
- preference : array-like, shape (n_samples,) or float, optional,近似数组,每个点的偏好 - 具有较大偏好值的点更可能被选为聚类的中心点。 簇的数量，即集群的数量受输入偏好值的影响。 如果该项未作为参数，则选择输入相似度的中位数作为偏好
- 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。
- verbose : boolean, optional, default: False

成员属性:

- cluster_centers_indices_ : array, shape (n_clusters,)聚类的中心索引,Indices of cluster centers
- cluster_centers_ : array, shape (n_clusters, n_features)聚类中心（如果亲和力=预先计算）。
- labels_ : array, shape (n_samples,)每个点的标签
- affinity_matrix_ : array, shape (n_samples, n_samples)存储拟合中使用的亲和度矩阵。
- n_iter_ : int,收敛的迭代次数。
- 近邻传播算法的复杂度是点数的二次。

成员方法:

- fit(X[, y]) 从负欧氏距离创建相似度矩阵，然后应用于近邻传播聚类。
- fit_predict 在 X 上执行聚类并返回聚类标签
- get_params 获取此估算器的参数
- predict(X) 预测 X 中每个样本所属的最近聚类
- set_params 设置此估算器的参数。
- __init__(damping=0.5, max_iter=200, convergence_iter=15, copy=True, preference=None, affinity='euclidean', verbose=False) 初始化函数

参考文献：https://blog.csdn.net/manjhOK/article/details/79586791

3. Mean-shift

参数：

- bandwidth=None： 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=None：array, shape=[n_samples, n_features], 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=False：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=1： int，clustering.get_bin_seeds 的参数，设置的最少质心个数；
  以上三个参数要结合起来理解；
- cluster_all=True： boolean，如果为 True，所有的点都会被聚类，包括不在任何核内的孤立点，其会选择一个离自己最近的核；如果为 False，孤立点的类标签为 -1；
- n_jobs=1： int，多线程；
  -1：使用所有的 cpu；
  1：不使用多线程；
  -2：如果 n_jobs<0，(n_cpus + 1 + n_jobs)个 cpu 被使用，所以 n_jobs=-2 时，所有的 cpu 中只有一块不被使用；

Mean Shift 算法通过更新质心的候选位置为所选定区域的偏移均值。然后，这些候选者在后处理阶段被过滤以消除近似重复，从而形成最终质心集合。

参考文献：https://blog.csdn.net/xiaoleiniu1314/article/details/80019182

4. Spectral clustering

谱聚类

n_clusters：代表我们在对谱聚类切图时降维到的维数（原理篇第 7 节的 k1k1），同时也是最后一步聚类算法聚类到的维数(原理篇第 7 节的 k2k2)。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。选择自定义相似矩阵时，需要自己调用 set_params 来自己设置相似矩阵。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。自定义核函数我没有使用过，这里就不多讲了。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。

核函数参数 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。

多项式核函数中这个参数对应 K(x,z)=（γx·z+r)dK(x,z)=（γx·z+r)d 中的 γγ。一般需要通过交叉验证选择一组合适的 γ,r,dγ,r,d

高斯核函数中这个参数对应 K(x,z)=exp(γ||x−z||2)K(x,z)=exp(γ||x−z||2)中的 γγ。一般需要通过交叉验证选择合适的 γγ

sigmoid 核函数中这个参数对应 K(x,z)=tanh（γx·z+r)K(x,z)=tanh（γx·z+r)中的 γγ。一般需要通过交叉验证选择一组合适的 γ,rγ,r
γγ默认值为 1.0，如果我们 affinity 使用'nearest_neighbors'或者是'precomputed'，则这么参数无意义。
核函数参数 degree：如果我们在 affinity 参数使用了多项式核函数 'poly'，那么我们就需要对这个参数进行调参。

参考文献：http://www.cnblogs.com/pinard/p/6235920.html

5. Ward hierarchical clustering

AgglomerativeClustering 参数说明:
AgglomerativeClustering(affinity='euclidean',
compute_full_tree='auto',
connectivity=None,
linkage='ward',
memory=Memory(cachedir=None),
n_clusters=6,
pooling_func=)

- affinity='euclidean'：距离度量方式
- connectivity：是否有连通性约束
- linkage='ward'：链接方式
- memory：存储方式
- n_clusters=6：簇类数

参考文献：https://blog.csdn.net/qq_39388410/article/details/78240037

6. Agglomerative clustering

同上，ward hierarchical clustering 是 Agglemerative linkage='ward'的特殊情况
我选定了 linkage='average'

7. DBSCAN

class sklearn.cluster.DBSCAN(eps=0.5,
min_samples=5,
metric='euclidean
', algorithm='aut
o', leaf_size=30,
p=None, n
_jobs=1)

- eps： DBSCAN 算法参数，即我们的 ϵϵ-邻域的距离阈值，和样本距离超过 ϵϵ 的样本点不在 ϵϵ-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 ϵϵ-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 过小的话，则会产生大量的核心对象，可能会导致类别数过少。
- metric：最近邻距离度量参数。可以使用的距离度量较多，一般来说 DBSCAN 使用默认的欧式距离（即 p=2 的闵可夫斯基距离）就可以满足我们的需求。
- algorithm：最近邻搜索算法参数，算法一共有三种，第一种是蛮力实现，第二种是 KD 树实现，第三种是球树实现。这三种方法在 K 近邻法(KNN)原理小结中都有讲述，如果不熟悉可以去复习下。对于这个参数，一共有 4 种可选输入，'brute'对应第一种蛮力实现，'kd_tree'对应第二种 KD 树实现，'ball_tree'对应第三种的球树实现， 'auto'则会在上面三种算法中做权衡，选择一个拟合最好的最优算法。需要注意的是，如果输入样本特征是稀疏的时候，无论我们选择哪种算法，最后 scikit-learn 都会去用蛮力实现'brute'。个人的经验，一般情况使用默认的 'auto'就够了。 如果数据量很大或者特征也很多，用"auto"建树时间可能会很长，效率不高，建议选择 KD 树实现'kd_tree'，此时如果发现'kd_tree'速度比较慢或者已经知道样本分布不是很均匀时，可以尝试用'ball_tree'。而如果输入样本是稀疏的，无论你选择哪个算法最后实际运行的都是'brute'。
- leaf_size：最近邻搜索算法参数，为使用 KD 树或者球树时， 停止建子树的叶子节点数量的阈值。这个值越小，则生成的 KD 树或者球树就越大，层数越深，建树时间越长，反之，则生成的 KD 树或者球树会小，层数较浅，建树时间较短。默认是 30. 因为这个值一般只影响算法的运行速度和使用内存大小，因此一般情况下可以不管它。
- p: 最近邻距离度量参数。只用于闵可夫斯基距离和带权重闵可夫斯基距离中 p 值的选择，p=1 为曼哈顿距离， p=2 为欧式距离。如果使用默认的欧式距离不需要管这个参数。

参考文献： *https://www.cnblogs.com/pinard/p/6217852.html*

8. **Gaussian mixtures**

```
GaussianMixture(n_components=1,
    covariance_type='full',
    tol=0.001,
    reg_covar=1e-06,
    max_iter=100,
    n_init=1,
    init_params='kmeans', weights_init=None, means_init=None,
    precisions_init=None,
    random_state=None,
    warm_start=False,
    verbose=0,
    verbose_interval=10)
```

参数：

- n_components:混合高斯模型个数，默认为 1
- covariance_type:协方差类型，包括{'full','tied', 'diag', 'spherical'}四种，分别对应完全协方

差矩阵（元素都不为零），相同的完全协方差矩阵（HMM 会用到），对角协方差矩阵（非对角为零，对角不为零），球面协方差矩阵（非对角为零，对角完全相同，球面特性），默认'full' 完全协方差矩阵

- tol：EM 迭代停止阈值，默认为 1e-3.
- reg_covar:协方差对角非负正则化，保证协方差矩阵均为正，默认为 0
- max_iter:最大迭代次数，默认 100
- n_init:初始化次数，用于产生最佳初始参数，默认为 1
- init_params: {'kmeans', 'random'}, defaults to 'kmeans'.初始化参数实现方式，默认用 kmeans 实现，也可以选择随机产生
- weights_init:各组成模型的先验权重，可以自己设，默认按照 7 产生
- means_init:初始化均值，同 8
- precisions_init:初始化精确度（模型个数，特征个数），默认按照 7 实现
- random_state :随机数发生器
- warm_start :若为 True，则 fit（）调用会以上一次 fit（）的结果作为初始化参数，适合相同问题多次 fit 的情况，能加速收敛，默认为 False。
- verbose :使能迭代信息显示，默认为 0，可以为 1 或者大于 1（显示的信息不同）
- verbose_interval :与 13 挂钩，若使能迭代信息显示，设置多少次迭代后显示信息，默认 10 次。

3 实验内容

1.1 实验流程

数据集包含 89 个类的共 2472 篇文档。

1. 首先，要构建每篇文档的文档向量。以下步骤是手动建立文档向量的过程，代码中也有这一部分。实际上也可以直接调用 sklearn 中的 TfidfVectorizer 来进行处理。
   1） 构建向量之前要对文档进行预处理，本次实验的数据已经经过预处理，每篇文档都只留下了关键词。
   2） 计算并存储 tf 值。注意每个词在每篇文档中都有一个 tf 值。
   3） 创建并存储词典。使用关键词构建词典。为了避免词典过大，在此我过滤掉了词频在 10 以下的低频词。这是合理的，因为词频非常低的时候，这个词是不具备区分文档类别的能力的。
   4） 计算并存储 idf 值。对于上一步创建的词典中的每个词，统计出现该词的训练集文档的数目，因此词典中的每个词都有一个 idf 值。
   5） 计算 tf x idf 的值并存储在文档向量矩阵中。此时的文档向量是依据之前创建的词典构建的，向量的长度--也就是之后的矩阵维度--就是词典的长度。矩阵长度就是文档个数。只需要计算词典中这些词的 tf x idf 值。
   6） 向量归一化。将所有的文档向量转化为单位向量。
2. 使用多种不同的聚类算法进行聚类。
3. 使用 sklearn 中的 NMI 对聚类结果进行评价。

3.2 实验环境

处理器：Intel(R) Core(TM) i7-8700K CPU @ 3.70Ghz 3.70Ghz
RAM：16.0 GB
系统： Windows 10，64 位
编程语言：Python 3.7
IDE: PyCharm

1.2 核心代码及注释

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.cluster import AffinityPropagation
from sklearn.cluster import MeanShift
from sklearn.cluster import SpectralClustering
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn import metrics

#####data process#######
f1=open("Tweets.txt")
tweets=f1.readlines()
numOfDocuments=len(tweets) # number of total documents in this
dataset
#print(numOfDocuments)
cate_tweets=[]#documents with the sign of cluster
for i in range(150):
    cate_tweets.append(0)

for tweet in tweets:
    splitTweet=tweet.split(":")
    body=splitTweet[1].split(",")
    content=body[0].strip('\"')
    content=content.strip(' \"')
    cluster=splitTweet[2].strip("\n")
    cluster=int(cluster.strip("}"))
    #print(cluster)
    if(cate_tweets[cluster]==0):
        cate_tweets[cluster]=[]
        cate_tweets[cluster].append(content)
    else:
        cate_tweets[cluster].append(content)
```

```
#print(cate_tweets)

pure_doc=[] #Pure documents without the sign of cluster
for i in range(len(cate_tweets)):
    if cate_tweets[i]==0:
        continue
    else:
        for doc in cate_tweets[i]:
            pure_doc.append(doc)

tfidf_vectorizer=TfidfVectorizer()

tfidf_matrix = tfidf_vectorizer.fit_transform(pure_doc)

#f2=open("tfidf_matrix_sklearn.txt","w")
#for item in tfidf_matrix:
    #f2.write("%s\n"%item)
#f2.close()

#####data process finished#######

#true labels#
ff=open("category.txt")
cate_content=ff.readlines()
category=[]
for cate in cate_content:
    category.append(int(cate.strip("\n")))
print(category)
labels_true=category

#kmeans#
num_clusters = 89
km_cluster = KMeans(n_clusters=num_clusters, max_iter=300,
n_init=40, init='k-means++', n_jobs=-1)
#labels generated by clustering#
result = km_cluster.fit_predict(tfidf_matrix)
labels_pred=result
f2=open("kmeans_cluster_sklearn.txt","w")
for res in result:
    f2.write("%s\n"%res)
f2.close()

#evaluation using NMI#
kmeans_score=metrics.normalized_mutual_info_score(labels_true,lab
```

```
els_pred)
print("Kmeans:%s"%kmeans_score)

#AffinityPropagation
ap_cluster=AffinityPropagation()
result=ap_cluster.fit_predict(tfidf_matrix)
labels_pred=result
f3=open("AffinityPropagation_cluster_sklearn.txt","w")
for res in result:
    f3.write("%s\n"%res)
f3.close()

#evaluation using NMI#
ap_score=metrics.normalized_mutual_info_score(labels_true,labels_p
red)
print("AffinityPropagation:%s"%ap_score)

#MeanShift
ms_cluster=MeanShift()
tfidf_matrix=tfidf_matrix.toarray()
result=ms_cluster.fit_predict(tfidf_matrix)
labels_pred=result
f4=open("MeanShift_cluster_sklearn.txt","w")
for res in result:
    f4.write("%s\n"%res)
f4.close()

#evaluation using NMI#
ms_score=metrics.normalized_mutual_info_score(labels_true,labels_p
red)
print("MeanShift:%s"%ms_score)

#SpectralClustering
sc_cluster=SpectralClustering()
result=sc_cluster.fit_predict(tfidf_matrix)
labels_pred=result
f5=open("SpectralClustering_cluster_sklearn.txt","w")
for res in result:
    f5.write("%s\n"%res)
f5.close()

#evaluation using NMI#
sc_score=metrics.normalized_mutual_info_score(labels_true,labels_pr
ed)
```

```
print("SpectralClustering:%s"%sc_score)

#Ward hierarchical clustering
n_clusters=89
wh_cluster=AgglomerativeClustering(n_clusters=n_clusters,linkage='
ward')
tfidf_matrix=tfidf_matrix.toarray()
result=wh_cluster.fit_predict(tfidf_matrix)
labels_pred=result
f5=open("WardHierarchical_cluster_sklearn.txt","w")
for res in result:
    f5.write("%s\n"%res)
f5.close()

#evaluation using NMI#
wh_score=metrics.normalized_mutual_info_score(labels_true,labels_p
red)
print("WardHierarchical:%s"%wh_score)


#AgglomerativeClustering
n_clusters=89
agg_cluster=AgglomerativeClustering(n_clusters=n_clusters,linkage=
"average")
tfidf_matrix=tfidf_matrix.toarray()
result=agg_cluster.fit_predict(tfidf_matrix)
labels_pred=result
f5=open("AgglomerativeClustering_cluster_sklearn.txt","w")
for res in result:
    f5.write("%s\n"%res)
f5.close()

#evaluation using NMI#
agg_score=metrics.normalized_mutual_info_score(labels_true,labels_
pred)
print("AgglomerativeClustering:%s"%agg_score)

#DBSCAN
db_cluster=DBSCAN(eps=1,min_samples=0.5)
result=db_cluster.fit_predict(tfidf_matrix)
labels_pred=result
f5=open("DBSCAN_cluster_sklearn.txt","w")
for res in result:
    f5.write("%s\n"%res)
```

```
f5.close()

#evaluation using NMI#
db_score=metrics.normalized_mutual_info_score(labels_true,labels_p
red)
print("DBSCAN:%s"%db_score)

#Gaussian Mixture
tfidf_matrix=tfidf_matrix.toarray()
gauss_cluster=GaussianMixture(n_components=89,covariance_type=
'full').fit(tfidf_matrix)

result=gauss_cluster.predict(tfidf_matrix)
labels_pred=result
f5=open("Gauss_cluster_sklearn.txt","w")
for res in result:
    f5.write("%s\n"%res)
f5.close()

#evaluation using NMI#
gauss_score=metrics.normalized_mutual_info_score(labels_true,label
s_pred)
print("Gauss:%s"%gauss_score)
```

4 实验结果

聚类方法	NMI 值
K-Means	0.784821989257618
Affinity propagation	0.783007621128966
Mean-shift	-0.7265625
Spectral clustering	0.44618130150558954
Ward hierarchical clustering	0.7800394104591925
Agglomerative clustering	0.8993232774941593
DBSCAN	0.7542496875513609
Gaussian mixtures	XXX

5 分析与讨论

MeanShift 的运行时间非常长，而且最后得到的 NMI 值不太正常。这里我还没有掌握调参的技巧。

DBSCAN 使用默认参数 NMI 值极低，大概在 0.2 左右。调整了 eps=1，min_samples=0.5 之后可以上升到 0.75.
Gaussian mixtures 方法在运行过程中出现了内存溢出现象。
回顾实验过程，发现我自己写的构建文档向量的代码，效果不如 sklearn 中自带的 TfidfVectorizer 效果好，可能和低频词的筛选的阈值等参数有关。