Q4（GMM）

姓名：王依睿

学号：1552651

GMM

1. 引入`sklearn`和`DBSCAN`

```
import sklearn
from sklearn.mixture import GaussianMixture
```

2. GMM聚类并计算其silhouette系数

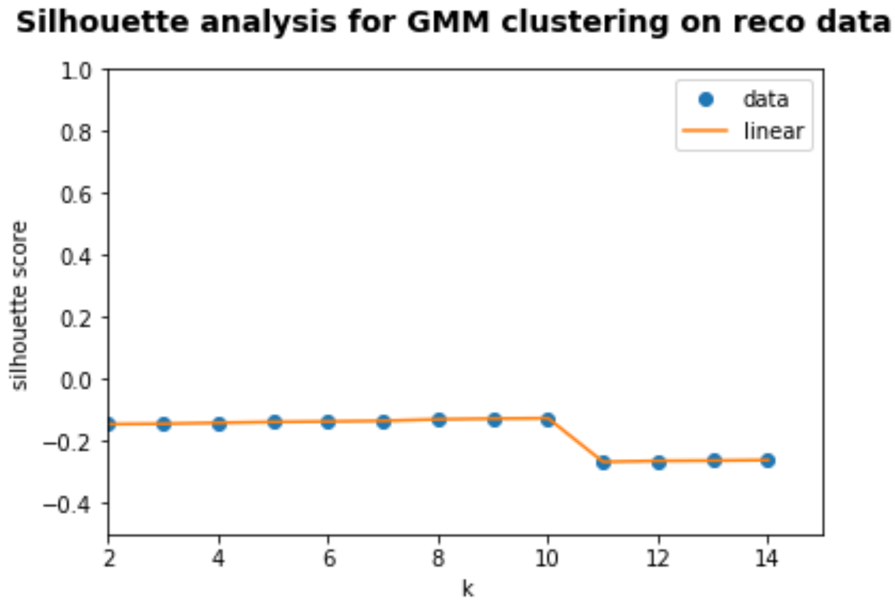
```
# 使用gmm进行聚类
def gmm(n_components):
    X = StandardScaler().fit_transform(data.T)
    gmm = GaussianMixture(n_components = n_components, covariance_type
= 'full', max_iter=20, random_state=0)
    gmm.fit(X)
    labels = gmm.predict(X)
    silhouette_score = metrics.silhouette_score(X, labels)
    return silhouette_score
```

3. 选择不同的n_components进行GMM聚类并计算其silhouette系数

```
# 选择不同的n_components聚类并计算其silhouette系数
def vip_gmm():
    vip_gmm_df = pd.DataFrame(columns = ['silhouette_score'], index =
np.arange(2, 15))
    for n_components in np.arange(2, 15):
        silhouette_score = gmm(n_components)
        vip_gmm_df['silhouette_score'][n_components] = silhouette_score
    return vip_gmm_df
```

结果

	silhouette_score
2	-0.145406
3	-0.144223
4	-0.141228
5	-0.138283
6	-0.136833
7	-0.134771
8	-0.129873
9	-0.127724
10	-0.126421
11	-0.26715
12	-0.264095
13	-0.26301
14	-0.261306



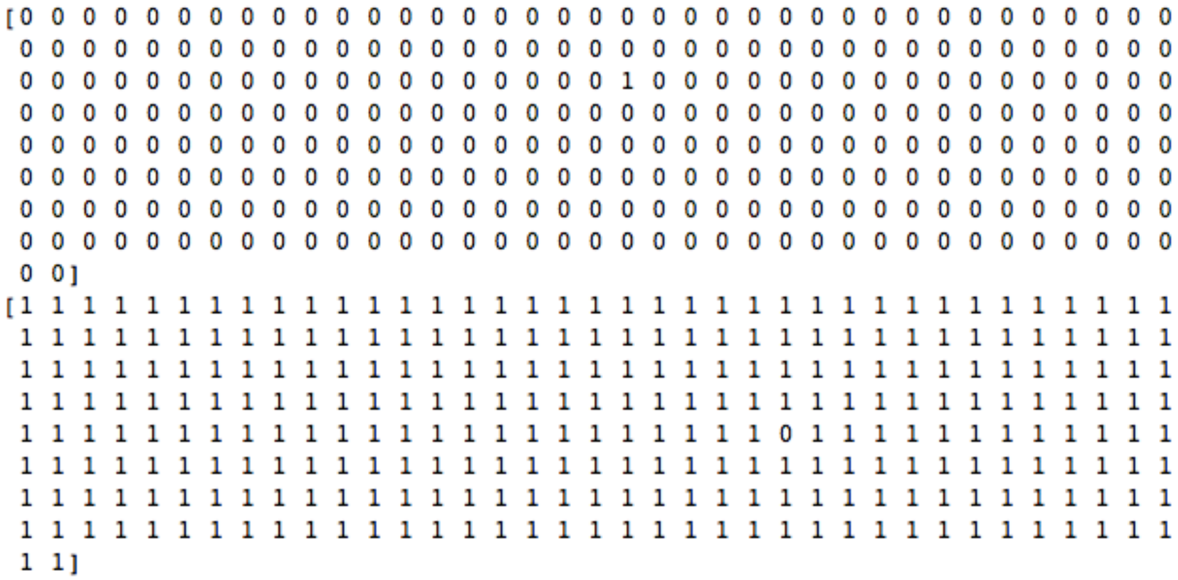
Silhouette analysis for GMM clustering on reco data

a) 假定以Kmeans作为真实的聚类结果

1. 选择Kmeans的最优聚类为2
2. 查看Kmeans和GMM的聚类结果

```
def gmm_kmeans(n_clusters):
    X = StandardScaler().fit_transform(data.T)
    kmeans_labels = KMeans(n_clusters = n_clusters).fit_predict(X)
    gmm = GaussianMixture(n_components = n_clusters, covariance_type =
'full', max_iter=20, random_state=0)
    gmm.fit(X)
    gmm_labels = gmm.predict(X)
    print(kmeans_labels)
    print(gmm_labels)
```

结果

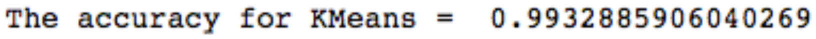


```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
```

2. 计算GMM的准确率(accuracy)

```
cnt = 0
    for i, gmm_label in enumerate(gmm_labels):
        if gmm_label != kmeans_labels[i]:
            cnt += 1
    print("The accuracy for KMeans = ", cnt / float(len(gmm_labels)))
```

结果



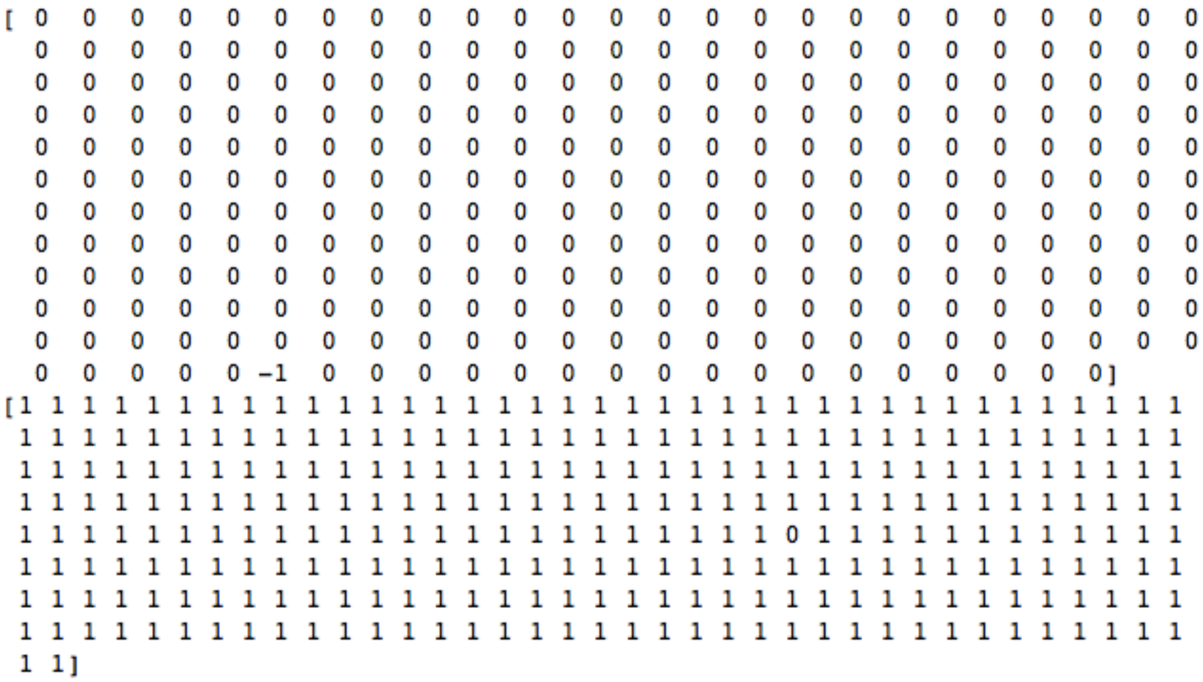
```
The accuracy for KMeans =  0.9932885906040269
```

b）比较DBScan和kmeans最优聚类

1. 选择DBScan的最优聚类为2
2. 查看DBScan和GMM的聚类结果

```
def dbscan_kmeans(n_clusters):
    X = StandardScaler().fit_transform(data.T)
    db = DBSCAN(eps = 130).fit(X)
    dbscan_labels = db.labels_
    gmm = GaussianMixture(n_components = n_clusters, covariance_type =
'full', max_iter=20, random_state=0)
    gmm.fit(X)
    gmm_labels = gmm.predict(X)
    print(dbscan_labels)
    print(gmm_labels)
```

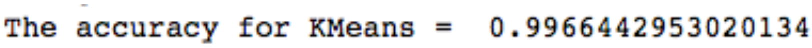
结果



```
[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
```

2. 计算GMM的准确率(accuracy)

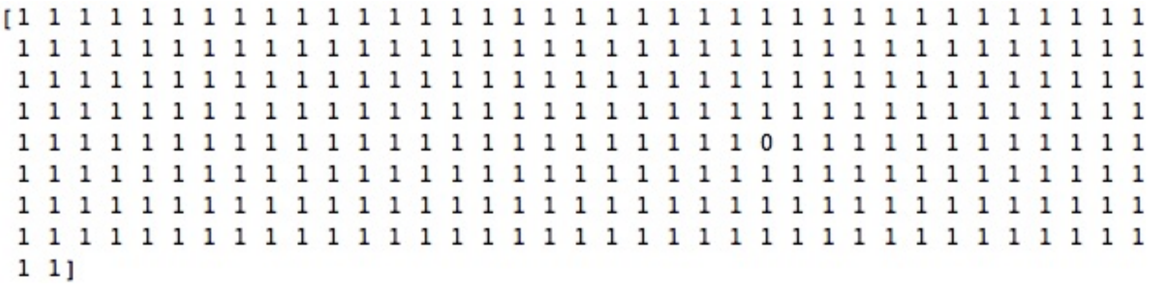
结果



```
The accuracy for KMeans =  0.9966442953020134
```

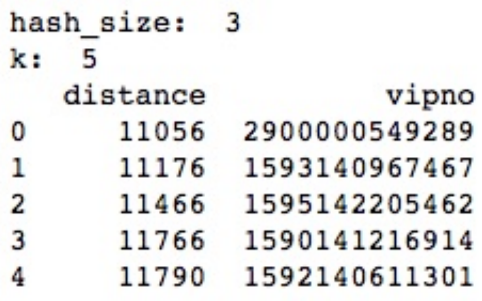
c) 验证lsh的knn查询结果

1. 查看GMM的聚类结果



```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
```

2. 查看knn的聚类结果



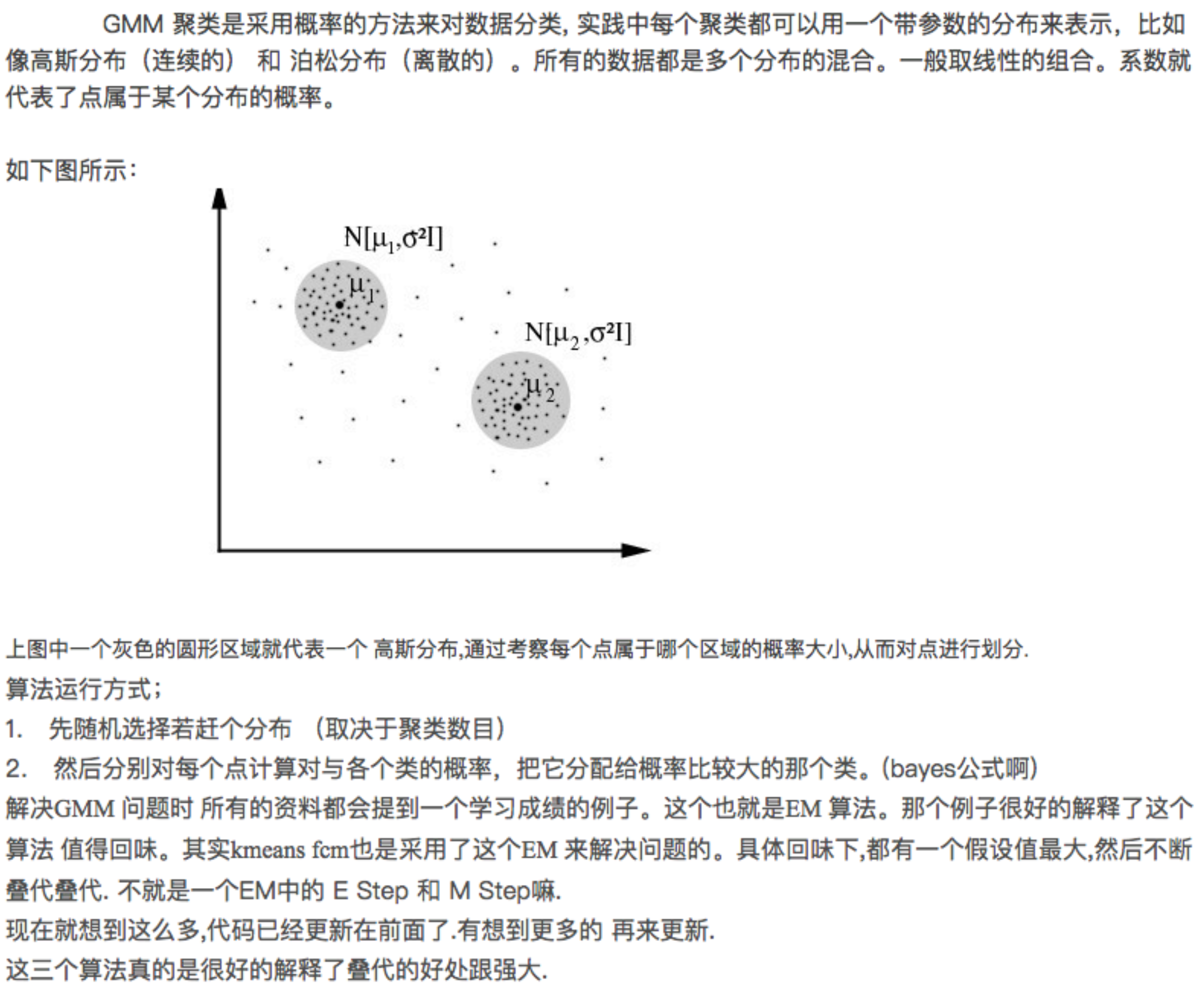
```
hash_size:   3
k:   5
    distance            vipno
0      11056   2900000549289
1      11176   1593140967467
2      11466   1595142205462
3      11766   1590141216914
4      11790   1592140611301
```

3. knn查询结果与输入vipno在同一个簇

讨论

1. 若将Silhouette系数作为评价GMM聚类质量的标准，Silhouette系数都比较低
2. 当聚类数目相等时，假定以Kmeans和DBScan作为真实的聚类结果，GMM的准确率都达到了99%以上，表明聚类结果非常相似。但当查看各种聚类算法结果时，都是极大多数点在同一类，只有1个点在另一类
3. 这次数据的特性使得通过聚类结果较难比较四种聚类算法的特性和优劣

GMM原理



GMM 聚类是采用概率的方法来对数据分类，实践中每个聚类都可以用一个带参数的分布来表示，比如像高斯分布（连续的） 和 泊松分布（离散的）。所有的数据都是多个分布的混合。一般取线性的组合。系数就代表了点属于某个分布的概率。

如下图所示：

上图中一个灰色的圆形区域就代表一个 高斯分布,通过考察每个点属于哪个区域的概率大小,从而对点进行划分.

算法运行方式；

1. 先随机选择若赶个分布 （取决于聚类数目）
2. 然后分别对每个点计算对与各个类的概率，把它分配给概率比较大的那个类。(bayes公式啊)

解决GMM 问题时 所有的资料都会提到一个学习成绩的例子。这个也就是EM 算法。那个例子很好的解释了这个算法 值得回味。其实kmeans fcm也是采用了这个EM 来解决问题的。具体回味下,都有一个假设值最大,然后不断叠代叠代. 不就是一个EM中的 E Step 和 M Step嘛.

现在就想到这么多,代码已经更新在前面了.有想到更多的 再来更新.

这三个算法真的是很好的解释了叠代的好处跟强大.