数据挖掘实验报告

聚类技术

姓　　名：汤凯(24320142202489)

王崇菲(24320142202492)

陈东东(24320142202402)

指导老师：刘昆宏

实验地点：海韵实验楼

完成时间：2017.10.28

一. 实验目的

使用 scikit-learn 包中的 kmeans, dbscan 等聚类算法分析数据。

比较不同参数对聚类结果的影响，使用不同测度分析聚类的质量，包括：

precision[i][j] = 预定义第 i 类并被分配到第 j 个聚类的样本数 / 第 j 个聚类中的样本数

recall[i][j] = 预定义第 i 类并被分配到第 j 个聚类的文档数 / 预定义第 i 类的样本数

f[i][j] = 2 * precision[i][j] * recall[i][j] / (precision[i][j] + recall[i][j])

二. 实验内容

熟悉 scikit-learn 包中的 kmeans, dbscan 等聚类算法，并能够熟练的调整算法参数，实现数据的预处理，并对结果进行分析。

三. 实验步骤以及结果

1、K-means 算法

（1）K-means 基本介绍：

K-means 算法接受输入量 k；然后将 n 个数据对象划分为 k 个聚类以便使得获得的聚类满足：：同一聚类中的对象相似度比较高；而不同聚类中的对象相似度较小。聚类相似是利用各聚类中对象的均值所获得一个“中心对象”（引力中心）来进行计算。

K-means 算法，主要有几个参数需要进行设置，我们实验主要采用的是调整三个参数（其中一个参数 k，由于我们的数据是给定的几类，也就是说我们的 k 值是确定的，所以我们就不对 k 进行调整）

K-means 算法我们采用的是如下初始化方法：

KMeans(init='k-means++', n_clusters=n_digits, n_init=100)

其中 init: 初始簇中心的获取方法；n_clusters:簇的个数，即你想聚成几类（也即我们上面说的 k）；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 个质心，实现算法，然后返回最好的结果。

（2）实验步骤：

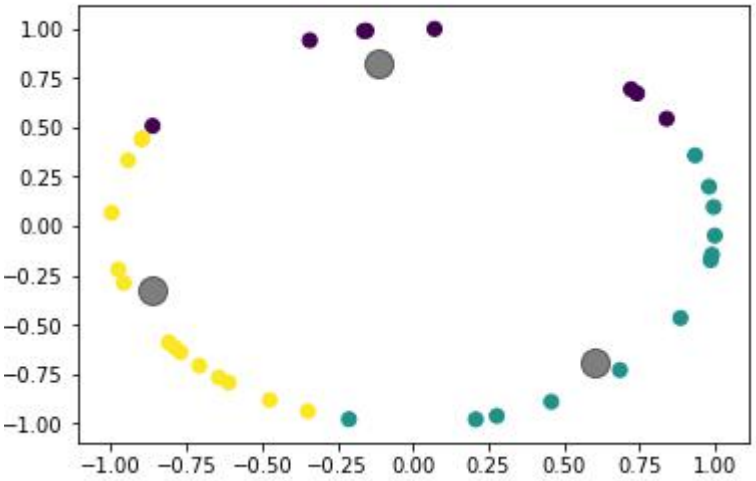
- 步骤一：数据读取以及处理
  - 首先是把我们的训练数据和测试数据读入（由于我们的数据是处理好的，所以我们没有进行缺失数据处理等处理，只是做了一个简单的读取）；

```
def get_kmeans_result(filename):
    #读取训练数据测试数据
    data, labels = read('data\\'+ filename +'_train.data')
    data_test , test_lable = read('data\\'+ filename +'_test.data');
```

  - 当然，由于我们想把数据结果可视化，所以我们也采用了第二种方法，那就是给数据降维，然后以图形的形式展示出来，所以这边也需要对数据进行进一步的处理

```
reduced_data = PCA(n_components=2).fit_transform(data)
reduced_data = normalize(reduced_data)
t0 = time()
kmeans_pca = KMeans(init='k-means++', n_clusters=n_digits, n_init=100)
t1 = time()
kmeans_pca.fit(reduced_data)
t2 = time()
predict_data = PCA(n_components=2).fit_transform(data_test)
predict_data = normalize(predict_data)
y_kmeans = kmeans_pca.predict(predict_data)
```

运行的结果图大致是这样的：



- 步骤二：初始化我们的 K-means 算法，这里，我主要是使用了三个参数，其中一个参数是我们训练数据已经给出来了，所以之后我们会对其中的两个参数（ 初始簇中心的获取方法，以及获取初始簇中心的更迭次数）进行调整对比；

```
t0 = time()
kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=100).fit(data)
t1 = time()
```

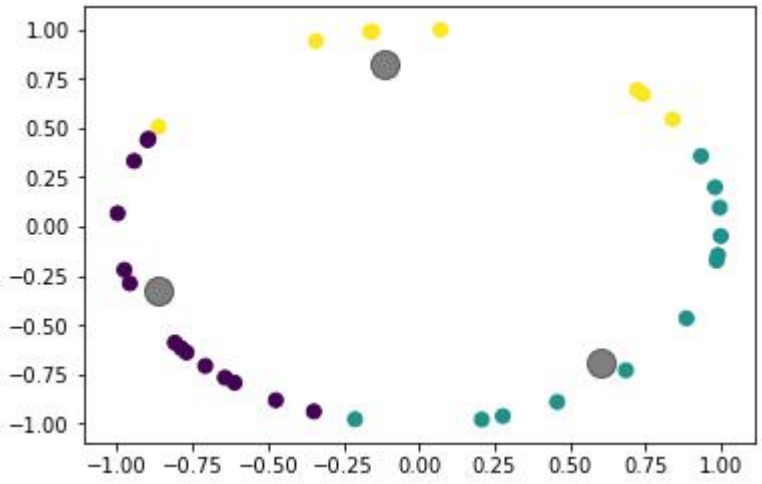
- 步骤三：接下来就是模型的训练和数据的测试了，由于这边是直接调用接口，所以简单的带过（这里有一个问题就是，如果初始方法的那两个参数是使用函数入参来初始化的情况下会很影响数据结果，很奇怪？）

```
kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=100).fit(data)
t1 = time()
predict_label = kmeans.predict(data_test)
t2 = time()
```

（3）实验结果对比（我们这边主要采用的是对其中一组测试数据 Leukemia1 进行分析）

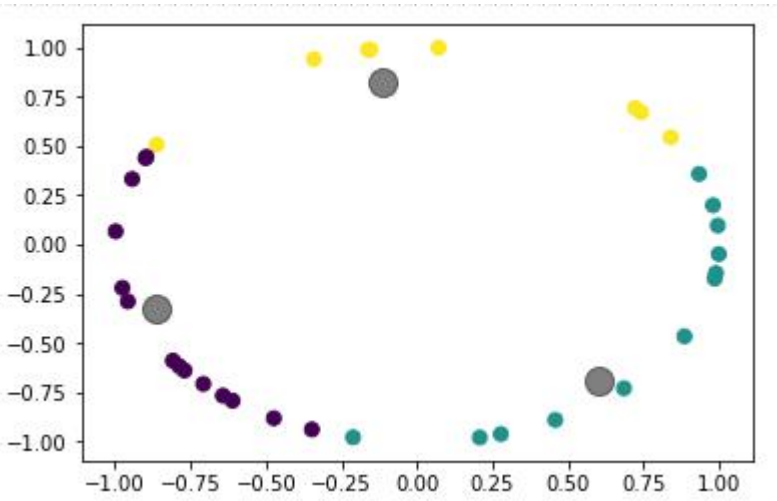
- init：初始簇中心的获取方法
  - 当使用“k-means++”来获取初始簇中心

```
未使用数据降维
n_digits: 3,     n_samples 38,   n_features 7129
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1113:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
precision: 0.41281512605
recall: 0.205882352941
fbeta_socre: 0.249087221095
模型训练时间： 3.9932281970977783
测试数据预测时间： 0.005000591278076172
数据降至二维维
n_digits: 3,     n_samples 38,   n_features 7129
precision: 0.771008403361
recall: 0.5
fbeta_socre: 0.604278074866
模型训练时间： 0.0
测试数据预测时间： 0.48102760314941406
```



■ 当使用“random”方式来获取初始簇中心时，结果如下：

```
未使用数据降维
n_digits: 3,     n_samples 38,   n_features 7129
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1113:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
precision: 0.790966386555
recall: 0.735294117647
fbeta_socre: 0.69887359199
模型训练时间： 3.5612032413482666
测试数据预测时间： 0.005000591278076172
数据降至二维维
n_digits: 3,     n_samples 38,   n_features 7129
precision: 0.771008403361
recall: 0.5
fbeta_socre: 0.604278074866
模型训练时间： 0.0
测试数据预测时间： 0.4450252056121826
```



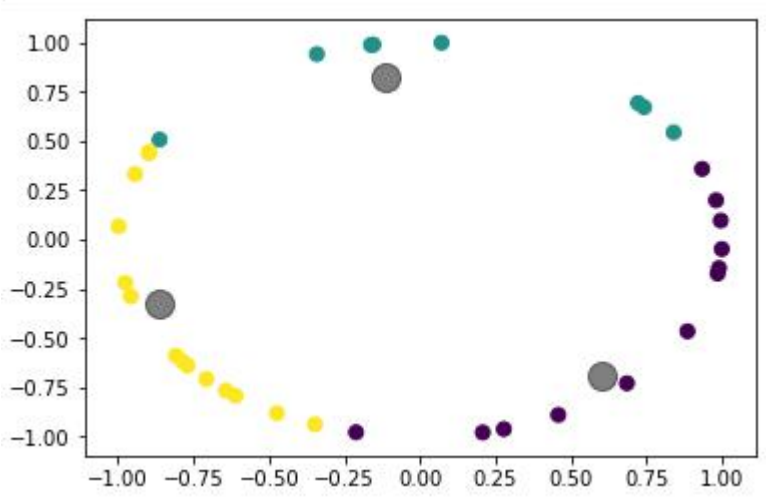
- 实验结果分析：

K-Means 算法需要用初始随机种子点来搞，这个随机种子点太重要，不同的随机种子点会有得到完全不同的结果。（K-Means++算法可以用来解决这个问题，其可以有效地选择初始点）

然而从实验结果来看，在不降维的情况下，不使用 K-means++选取初始随机种子，我们的聚类结果是比较好的，而使用了 K-means++反而有点不尽人意。同样，对于我们降维之后的数据，虽然两种方法所得到的结果差不多，但是没有使用 K-means++的效果要更好点，包括在性能和时间上。

- n_init: 获取初始簇中心的更迭次数（我们这里主要是采用两种，一个是迭代 10 次，一个是迭代 100 次，而 100 次结果在上图可见，我们这边 init 尽量采用“random”的方法）
  - 迭代 10 次，即 n_init=10

```
未使用数据降维，init='random'，n_init=10
n_digits: 3,     n_samples 38,   n_features 7129
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1113:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
precision: 0.353921568627
recall: 0.558823529412
fbeta_socre: 0.43337334934
模型训练时间： 0.3950226306915283
测试数据预测时间： 0.004000186920166016
数据降至二维，init='random'，n_init=10
n_digits: 3,     n_samples 38,   n_features 7129
precision: 0.583333333333
recall: 0.411764705882
fbeta_socre: 0.478178368121
模型训练时间： 0.0
测试数据预测时间： 0.03200173377990723
```



- 实验结果分析：

  从结果来看，不论是在数据降维后还是没有降维，当迭代次数 n_init 降下来之后，我们的模型训练的结果都有所变差，但是相对来说，它的运行时间是更小的

- 数据降维和数据不降维之间的对比：

  从上述实验截图来看，我们没有使用数据降维的它的性能波动性比较大，而且在使用“random”初始化，并且迭代 100 次的时候效果是最佳的，但是降维之后的数据就相对比较稳定。

  但时总体上来讲，降维之后的数据训练时间以及测试时间都是比较好的。

2、DBScan 算法

（1）DBScan 基本介绍：

DBSCAN(Density-Based Spatial Clustering of Applications with Noise)是一个比较有代表性的基于密度的聚类算法。与划分和层次聚类方法不同，它将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇，并可在噪声的空间数据库中发现任意形状的聚类。

其核心思想使用一个点的邻域内的邻居点数衡量该店所在空间的密度。它可以找出形状不规则的 cluster，且聚类是不需要事先知道 cluster 的个数。

DBSCAN 算法中最重要的有两个参数：eps 和 min_samples，前者为定义密度时的邻域半径，后者为定义核心点时的阈值。

在以下的实验中，我们也将围绕这两个参数进行调整。

（2）实验步骤

- 步骤一：数据读取以及处理
  - 首先是把我们的训练数据（由于我们的数据是处理好的，所以我们没有进行缺失数据处理等处理，只是做了一个简单的读取），但是这边会存在一个问题就是，如果不对我们的数据进行一个降维并归一化处理的话，对于 eps 就不好设定，因为我们的数据的维度都是近万的，所以半径会很长的庞大，如果我们只是使用简单的 eps=0.5 的话就是什么结果都得不到，因为所有的点都变成了干扰点，结果如下；

```
数据未降维 eps: 0.5   min_samples:  3
precision: 0.0
recall: 0.0
fbeta_socre: 0.0
所用时间:  0.029001712799072266
```

- 基于上述存在的问题，所以我把数据进行了降维并归一化处理，另外就是由于我们的聚类结果是以数字的形式呈现的，所有我们在处理标签的时候也有吧标签转化为数字，方便后面的计算。

```
def get_DBSCAN_pca_result(filename, eps, min_samples):
    #读取训练数据测试数据
    data, labels = read('data\\'+ filename +'_train.data')
    #将数据降至二维
    reduced_data = PCA(n_components=2).fit_transform(data)
    reduced_data = normalize(reduced_data)
    #print(reduced_data)
    #data test , test lable = read('data\\'+ filename +' test.dat
```

标签转换函数：

```
def change_labels_2_num(labels):
    labels_dict = {}
   # print(labels)
    k = 0
    for i in labels:
        if i not in labels_dict.keys():
            labels_dict[i] = k
            k += 1
    #print(labels_dict)
    for i in range(len(labels)):
       labels[i] = labels_dict[labels[i]]
    #print(labels)
    return labels
```

- 步骤二：由于我们的数据进行过处理，然后我们主要还就是在 eps 和 min_samples 两个参数的调整上，为了使用的便利，我们 eps 采用了 3 个数值 0.25,0.5 以及 0.75，min_samples 也同样采用了 3 个数值 3,5 以及 7，我们使用这两组数据进行组合得到我们的实验结果：

```
get_DBSCAN_result('Leukemia1', 0.5, 3)
eps_s = [0.25, 0.5, 0.75]
min_samples_s = [3, 5, 7]
for eps in eps_s:
    for min_sample in min_samples_s:
        get_DBSCAN_pca_result('Leukemia1', eps, min_sample)
```

（3）实验结果对比

- 首先，我们先把实验的结果图粘贴出来：

```
数据降至二维 eps: 0.25          min_samples:  3
precision: 0.410495936812
recall: 0.263157894737
fbeta_socre: 0.318421052632
所用时间:  0.0
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1113: UndefinedMet
score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1115: UndefinedMet
are ill-defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1115: UndefinedMet
are ill-defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
数据降至二维 eps: 0.25          min_samples:  5
precision: 0.38543328017
recall: 0.236842105263
fbeta_socre: 0.292355889724
所用时间:  0.0010001659393310547
数据降至二维 eps: 0.25          min_samples:  7
precision: 0.447368421053
recall: 0.210526315789
fbeta_socre: 0.276518218623
所用时间:  0.0009999275207519531
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1115: UndefinedMet
are ill-defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1113: UndefinedMet
score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1115: UndefinedMet
are ill-defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
数据降至二维 eps: 0.5          min_samples:  3
precision: 0.643274853801
recall: 0.631578947368
fbeta_socre: 0.637017822776
所用时间:  0.0009999275207519531
```

```
数据降至二维 eps:C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1115: Undefir
Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
 0.5     min_samples:  5
precision: 0.640643274854
recall: 0.605263157895
fbeta_socre: 0.621980228791
所用时间:  0.0
数据降至二维 eps: 0.5          min_samples:  7
precision: 0.640643274854
recall: 0.605263157895
fbeta_socre: 0.621980228791
所用时间:  0.0
数据降至二维 eps:C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1115: Undefir
Recall and F-score are ill-defined and being set to 0.0 in labels with no true samples.
  'recall', 'true', average, warn_for)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1113: UndefinedMetricWarning:
score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
 0.75    min_samples:  3
precision: 0.25
recall: 0.5
fbeta_socre: 0.333333333333
所用时间:  0.0009999275207519531
数据降至二维 eps: 0.75         min_samples:  5
precision: 0.25
recall: 0.5
fbeta_socre: 0.333333333333
所用时间:  0.0
数据降至二维 eps: 0.75C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1113:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predi
  'precision', 'predicted', average, warn_for)
       min_samples:  7
precision: 0.25
recall: 0.5
fbeta_socre: 0.333333333333
所用时间:  0.0009999275207519531
```

- 从实验结果来看，当 eps=0.5，min_samples=3 的时候所得到的结果是最好的。

```
数据降至二维 eps: 0.5          min_samples:  3
precision: 0.643274853801
recall: 0.631578947368
fbeta_socre: 0.637017822776
所用时间:  0.0009999275207519531
```

其实每组参数所使用的时间基本上是差不多的。

四．总结

从以上的结果以及查阅资料来分析，我们可以很明显的得到如下结果：

1、kmeans 聚类的特点是：

（1）人为的输入要聚的类数 k（实验中，是直接根据我们的数据来定的）

（2）一般是计算的欧式距离判断相似性

（3）每次随机的选取 k 个聚类中心，聚类结果受随机选取的类中心影响比较大

（4）简单

（5）算法过程：

- 输入训练数据集，类别 K
- 随机的选取 K 条数据，作为 K 个类的中心
- 计算所有数据到 2 中的 K 个类中心的距离
- 根据 3 的结果，与某个类最近的数据化为一类
- 根据 4，从新得到 K 个类，并计算 K 个类的中心
- 更新上面的的过程，直到 K 类数据不再变化或者到达迭代次数位置

2、dbscan 是一种基于密度的聚类算法 ，与 kmeans 聚类相比：

（1）dbscan 可以发现任意形状的数据集
（2）且不用输入类别数 K
（3）核心是：如果一个点，在距它 Eps 的范围内有不少于 MinPts 个点，则该点就是核心点。核心和它 Eps 范围内的邻居形成一个簇。在一个簇内如果出现多个点都是核心点，则以这些核心点为中心的簇要合并。这样再逐步扩大，形成一个类也就是簇。
（4）dbscan 聚类不仅能够发现核心点还能够找到边界点和噪声点，边界点是属于某个类的边界点，噪声点不属于任何一类。我们实验中，对于噪声点的结果就是以-1 的形式展现出来的。
3、单纯的从实验结果来看，两种方法，我们的数据都是使用的 Leukemia1，这个数据，首先我们就来对比一下我们的实验数据的性能：

对于 kmeans 算法来讲，最好的实验效果是：

```
未使用数据降维
n_digits: 3,     n_samples 38,   n_features 7129
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1113:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in
labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
precision: 0.790966386555
recall: 0.735294117647
fbeta_socre: 0.69887359199
模型训练时间:  3.5612032413482666
测试数据预测时间:  0.005000591278076172
```

而对于 DBSCAN 算法，最好的实验效果是：

```
数据降至二维 eps: 0.5          min_samples:  3
precision: 0.643274853801
recall: 0.631578947368
fbeta_socre: 0.637017822776
所用时间:  0.0009999275207519531
```

我们可以很明显的发现，kmeans 的性能在我们实验数据的表现是相对来说表现比较出色的，但是相对的来讲，花的时间也就相对多一点。另外从算法对参数的依赖来讲，kmeans 算法在不同的参数下，结果的波动性不是特别的，但是对于我们的 DBSCAN 来讲，可能浮动就会相对比较大一点。

所以，对于我们的实验数据来讲，K-means 算法还是比较良好的。