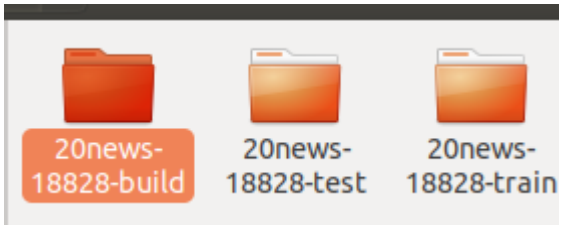
实验步骤：将数据集解压后，将其中的 80%作为训练集，20%作为测试集。



预处理时报错，缺少所需的 stopwords：

```
liubuntu@liubuntu:~/Desktop$ python knn.py
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/nltk/corpus/util.py", line 80, in __load
    try: root = nltk.data.find('{}/{}'.format(self.subdir, zip_name))
  File "/usr/local/lib/python3.6/dist-packages/nltk/data.py", line 675, in find
    raise LookupError(resource_not_found)
LookupError:
**********************************************************************
  Resource stopwords not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('stopwords')

  Searched in:
    - '/home/liubuntu/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
    - '/usr/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/lib/nltk_data'
**********************************************************************
```

解决办法：

```
liubuntu@liubuntu:~/Desktop$ python knn.py
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/liubuntu/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

python 读取文件时发生了错误

```
    main()
  File "knn.py", line 199, in main
    preProcess()
  File "knn.py", line 25, in preProcess
    preProcessFile(dirs[i],files[j])
  File "knn.py", line 34, in preProcessFile
    data = open(src,"r").readlines()
  File "/usr/lib/python3.6/codecs.py", line 321, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xfd in position 293: invalid start byte
```

解决办法：

将

```
data = open(src,"r").readlines()
```

改为：

```
data = open(src,"rb").readlines()
```

单个文件的 VSM 结果：

运行结果：

```
liubuntu@liubuntu:~/Desktop$ python knn.py
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/liubuntu/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
correctness is  0.7659574468085106
```

代码原理：

1、 预处理：

```
#创建新文件夹，存放预处理后的文本数据
def preProcess():
    dirs=os.listdir(rootDir)
    for i in range(len(dirs)):
        if i == 0:
            continue
        fileDir=rootDir+dirs[i]
        buildFileDir=buildDir+dirs[i]

        files=os.listdir(fileDir)

        if os.path.exists(buildFileDir)==False:
            os.mkdir(buildFileDir)

        for j in range(len(files)):
            preProcessFile(dirs[i],files[j])

#建立目标文件夹，生成目标文件
def preProcessFile(dir,fileName):
    src = rootDir+ "/" + dir + "/" + fileName
    dst = buildDir + "/" + dir + "/" + fileName

    dstFile = open(dst,'w')
    data = open(src,"rb").readlines()

    for line in data:
        result = processLine(line)
        for word in result:
            dstFile.write('%s/n' % word)
```

```
            dstFile.close()



        #对每行字符串进行处理，主要是去除非字母字符，转换大写为小写，去除停用词
        def processLine(line):
            stopwords = nltk.corpus.stopwords.words('english') #去停用词
            porter = nltk.PorterStemmer()  #词干分析
            splitter = re.compile('[^a-zA-Z]')  #去除非字母字符，形成分隔
    for i in range(len(line)):
                if chr(line[i]).isalpha():
                string+=chr(line[i])
            else:
                string+=" "

            words = [porter.stem(word.lower()) for word in splitter.split(string)/
                if len(word)>0 and/
                word.lower() not in stopwords]
        return words

2、 建立 vsm
# 统计每个词的总的出现次数
def countWords():
    wordMap = {}
    wordMap2 = {}

    dirs = os.listdir(buildDir)
    for i in range(len(dirs)):
        filedir = buildDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            for line in open(filePath).readlines():
                word=line.strip('/n')
                wordMap[word]=wordMap.get(word,0) + 1
    for key, value in wordMap.items():
        if value > 10:#去掉出现次数小于 10 的词
            wordMap2[key]=value

    return wordMap2


#计算 idf
```

```
def IDF():
    wordFileMap = {}
    IDFMap = {}
    dirs = os.listdir(buildDir)
    for i in range(len(dirs)):
        filedir = buildDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            for line in open(filePath).readlines():
                word=line.strip('/n')

                if wordFileMap.get(word)== None:#该词在所有文件里第一次出现
                    wordFileMap[word]=[]
                    wordFileMap[word].append(filePath)
                elif filePath in wordFileMap[word] == False: #该词在该文件里第一次出现
                    wordFileMap[word].append(filePath)


    for word in wordFileMap:
        IDFMap[word] = 20000.0/len(wordFileMap[word]) #文件总数/该词出现的文件个数
    return IDFMap

#计算一个文件的向量
def Vector(file,wordMap,IDFMap):
    fileVector = {}
    for line in open(file).readlines():
        word=line.strip('/n')
        if word not in wordMap:
            continue
        if word not in fileVector:
            fileVector[word]=1
        else:
            fileVector[word]+=1

    for key in fileVector:
        fileVector[key]=fileVector[key]*IDFMap[word]
       return fileVector
```

3、 KNN 分类器

#建立训练集的 map{向量：分类}

```
def vectorMap(wordMap,IDFMap):
    vectorMap={}
    dirs = os.listdir(buildDir)
    for i in range(len(dirs)):
        filedir = buildDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            fileVector = Vector(filePath,wordMap,IDFMap)
            vectorMap[fileVector]=filedir

    return vectorMap

#计算两个向量的相似度
def similarity(trainVector,testVector):
    trainlist = []
    testlist = []
    for word in testVector:
        if word in trainVector:
            trainlist.append(trainVector[word])
            testlist.append(testlist[word])

    trainVect = numpy.mat(trainlist)
    testVect = numpy.mat(testlist)

    num = float(testVect * trainVect.T)
    denom = numpy.linalg.norm(testVect) * numpy.linalg.norm(trainVect)
    return float(num)/(1.0+float(denom))

#根据模型预测测试文件的分类
def testFileSimilarity(testFile,wordMap,IDFMap,trainVectors):
    similarityMap = {}
    testVectoer = Vector(testFile,wordMap,IDFMap)
    resultMap = {}
    for vector in trainVectors:
        sim = similarity(vector,testVectoer)
        similarityMap[sim] = trainVectors[vector]

    keys = similarityMap.keys()
    keys.sort()
    sortedResult =  [(k,similarityMap[k]) for k in sorted(similarityMap.keys())]
    count = 0

    for i in sortedResult:
```

```
        if i[1] not in resultMap:
            resultMap[i[1]] = 1
        else:
            resultMap[i[1]] += 1
        count = count + 1
        if count > 100:#选择的 K 值为 100
            break

    max = 0
    doctype = ""
    for key,value in resultMap:
        if value>max:
            doctype = key
            max = value

        return doctype
```

4、观察在测试集上运行的准确率

```
#预测文件的分类并计算正确度
def predictTestFileCorr(wordMap,IDFMap,trainVectors):
    dirs = os.listdir(testDir)

    for i in range(len(dirs)):
        filedir = testDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            fileNumber = fileNumber + 1
            predict = testFileSimilarity(filePath,wordMap,IDFMap,trainVectors)
            if predict == dirs[i]:
                correctCount = correctCount + 1

    return float(correctCount/fileNumber)

def main():
        preProcess()
        wordMap = countWords()
        IDFMap= IDF()
        vectors = vectorMap(wordMap,IDFMap)
        correctness = predictTestFileCorr(wordMap,IDFMap,vectors)
        print(correctness)
```

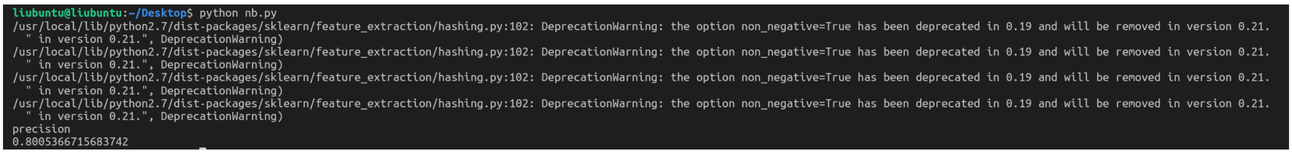
Homework2：Naive Bayes

代码及原理：

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.datasets import fetch_20newsgroups_vectorized
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics

def main():
    categories = ['comp.graphics',
    'comp.os.ms-windows.misc',
    'comp.sys.ibm.pc.hardware',
    'comp.sys.mac.hardware',
    'comp.windows.x',
    'rec.autos',
    'rec.motorcycles',
    'rec.sport.baseball',
    'rec.sport.hockey',
    'sci.crypt',
    'sci.electronics',
    'sci.med',
    'sci.space',
    'misc.forsale',
    'talk.politics.misc',
    'talk.politics.guns',
    'talk.politics.mideast',
    'talk.religion.misc',
    'alt.atheism',
    'soc.religion.christian'];

   #下载数据
    newsgroup_train = fetch_20newsgroups(subset = 'train',categories = categories);
    newsgroups_test = fetch_20newsgroups(subset = 'test',
                                         categories = categories);

  #将数据向量化
    vectorizer = HashingVectorizer(stop_words = 'english',non_negative = True,
                                   n_features = 10000)
    fea_train = vectorizer.fit_transform(newsgroup_train.data)
    fea_test = vectorizer.fit_transform(newsgroups_test.data);
    #创建朴素贝叶斯分类器
    clf = MultinomialNB(alpha = 0.01)
    clf.fit(fea_train,newsgroup_train.target);
    #用朴素贝叶斯分类器预测测试集
    pred = clf.predict(fea_test);
   #计算结果
    calculate_result(newsgroups_test.target,pred);
```

```
def calculate_result(actual,pred):
    m_precision = metrics.precision_score(actual,pred,average="weighted");
    print "precision"
    print m_precision

if __name__ == "__main__":
    main()
```

根据运行结果观察到准确率为：0.8005366715683742



实验报告三

一、实验要求

本次实验要求在 tweets 数据集上进行各种聚类算法的实验,一共包括八种聚类方式,然后用 NMI(Normalized Mutual Information)作为评价指标对聚类结果进行评价。

二、数据处理

本实验的数据是用 json 模式存储的,所以在解析的时候,可以用 python 自带的 json 模块进行解析。

三、NMI(Normalized Mutual Information)

熵的定义为:$H(X) = -\sum_{i=1}^{n} P(x_i)\log_b P(x_i)$;

$p(x,y)$

离散变量的互信息定义为:$I(X, Y) = \sum_{y\in Y}\sum_{x\in X} p(x, y)\log\left(\frac{p(x)p(y)}{}\right)$,

$I(X,Y)$

归一化互信息的定义为:$U(X, Y) = 2\ H(X)+H(Y)$

四、聚类算法

(一)、K-means

1、 算法原理

K-means 算法是硬聚类算法,是典型的基于原型的目标函数聚类方法的代表,它是数据点到原型的某种距离作为优化的目标函数,利用函数求极值的方法得到迭代运算调整规则。

2、 算法步骤

a) 随机选择 K 个随机的点,成为聚类质心;

b) 对剩余数据点计算它到每个质心的距离,并把它归到最近的质心的类;

c) 重新计算已经得到的各个类的质心;

d) 重复 a),b),c)直到新的质心与原质心相等或距离小于指定阈值;

3、 参数分析

a) K:表示聚类个数,不同的聚类个数会影响聚类效果;

(二)、Affinity Propagation

1、 算法原理

AP 算法的基本思想是将全部样本看作网络的节点,然后通过网络中各条边的消息传递计算出个样本的聚类中心。聚类过程中共有两种消息在各节点间传递,分别是吸引度和归属度,直到产生 m 个高质量的 exemplar,同时将剩余的数据点分配到相应的聚类中。2、 算法步骤

a) 计算初始的相似度矩阵,将各点之间的吸引度和归属度初始化为 0;

b) 更新各点之间的吸引度,随之更新各点之间的归属度;

c) 确定当前样本的代表样本点 k;

d) 重复 b),c),直到所有的样本的所属不再发生变化为止;

3、 参数分析

该聚类算法不需要额外设定参数,因为不需要事先指定聚类的数量,而且聚类的结果不会发生变化;

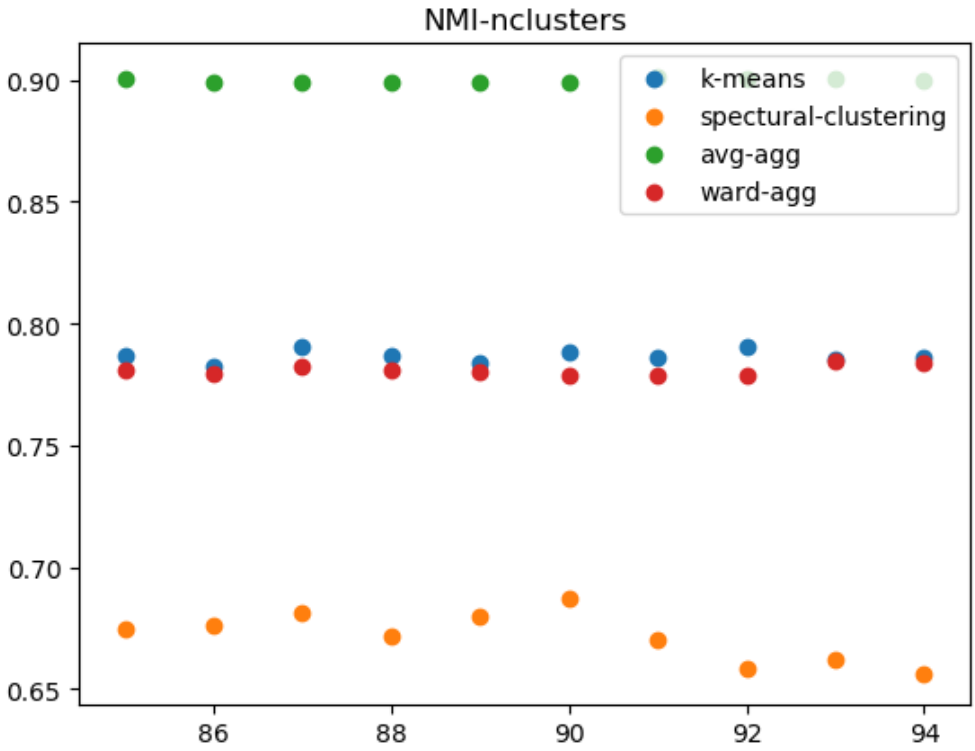
(三)、Spectral Clustering

1、 算法原理

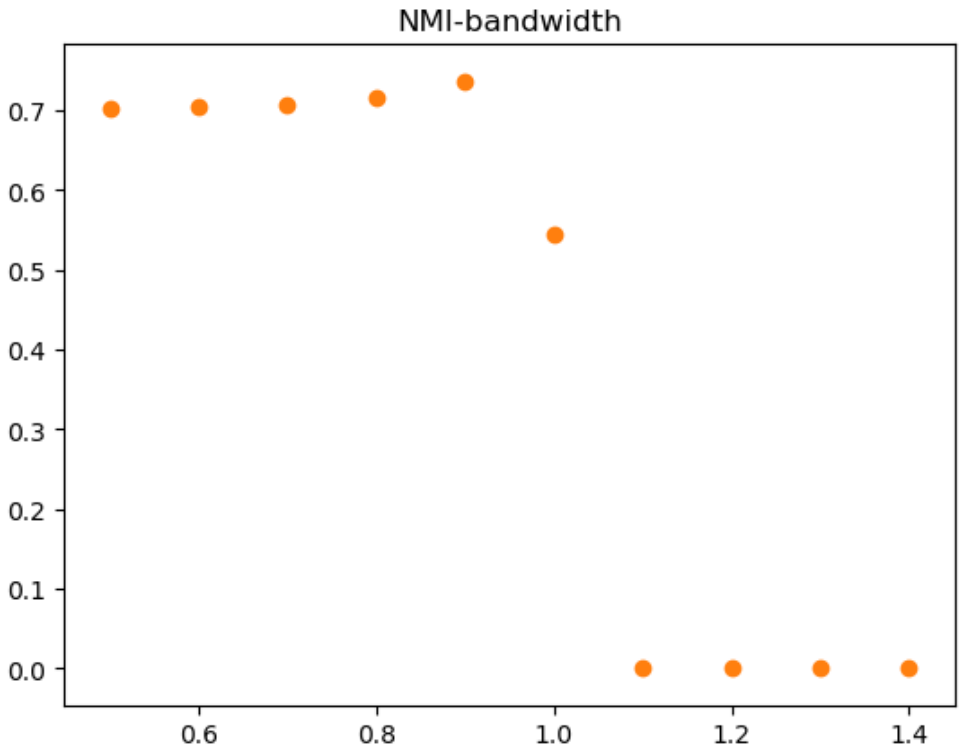
谱聚类是一种基于图论的聚类算法,主要思想是把所有的数据看成空间中的点,这些点之间可以用边连接起来。距离较远的两个点之间的边权重值较低,而距离较近的两个点之间的权重值较高,通过对所有数据点组成的图进行切图,让切图后不同的子图间边权重和尽可能的低,而子图内的边权重和尽可能的高,从而达到聚类的目的。

2、 算法步骤
a) 对样本构建相似度矩阵 S;
b) 根据相似度矩阵 S 构建邻接矩阵 W、度矩阵 D
c) 计算出拉普拉斯矩阵 L
d) 对拉普拉斯矩阵 L 进行标准化
e) 计算最小的 K 个特征值所对应的特征向量
f) 特征向量标准化,并组成特征矩阵 F
g) 将特征向量按照某种聚类方式聚类
3、 参数分析
a) K:聚类个数
(四)、Agglomerative Clustering
1、 算法原理
Agglomerative Clustering 是一种自底而上的层次聚类方法,可以在不同的层次上对数据集进行划分,形成树状的聚类结构。
2、 算法步骤
a) 将每个样本都作为一个簇;
b) 计算聚类簇之间的距离,找出距离最近的两个簇,将这两个簇合并
c) 重复 b),直到聚类簇的数量为 K
3、 参数分析
a) n_clusters: 整数,代表聚类个数
b) affinity: 一个字符串或者可调用对象,用于计算距离,可以为:“euclidean”, “L 1 ”, “L 2 ”, “manhattan”, “cosine”, 或 ‘precomputed’。但是作为字符串使用的时候,只有” Euclidean”可用!
c) linkage:一个字符串,用于指定链接算法,有三种方式:
i.
‘ward’:单链接 single-linkage,采用 d min ;
ii.
‘complete’:全链接 complete-linkage 算法,采用 d max ;
iii.
‘average’:均连接 average-linkage 算法,采用 d avg ;(五)、Mean Shift 聚类算法
1、 算法原理
Mean shift 算法是基于核密度估计的爬山算法;简单的说,mean shift 就是沿着密度上升的方向寻找同属一个簇的数据点;具体来说就是要定义一个均值漂移,然后不断迭代这个均值漂移的过程。
a) 均值漂移:给定 d 维空间的 n 个数据点集 X,那么对于空间中的任意点 x 的 mean shift 向量基本形式可以表示为: $M_h = K \sum x_i \in S_k (x_i - x)$ 这个向量就是漂移向量,其中 S_k 表示数据集内到 x 的距离小于圆的半径 h 的数据点。
而漂移过程就是利用漂移向量更新球心 x 的位置:$x = x + M_h$ 。
2、 聚类流程
a) 在未被标记的数据点中随机选择一个点作为 center;
b) 找出离 center 距离在 bandwidth 之内的所有点,记作集合 M,认为这些点属于族 c,同时把集合 M 内的点属于簇 c 的频率加 1,这个参数将用于最后步骤的分类;
c) 以 center 为中心,计算漂移向量 shift;
d) center= center+ shift,即将 center 按照 shift 进行移动,并将得到的新的 center 作为中心;
e) 重复 b),c),d),直至收敛,该过程中遇到的所有的点都应被标记为 c;

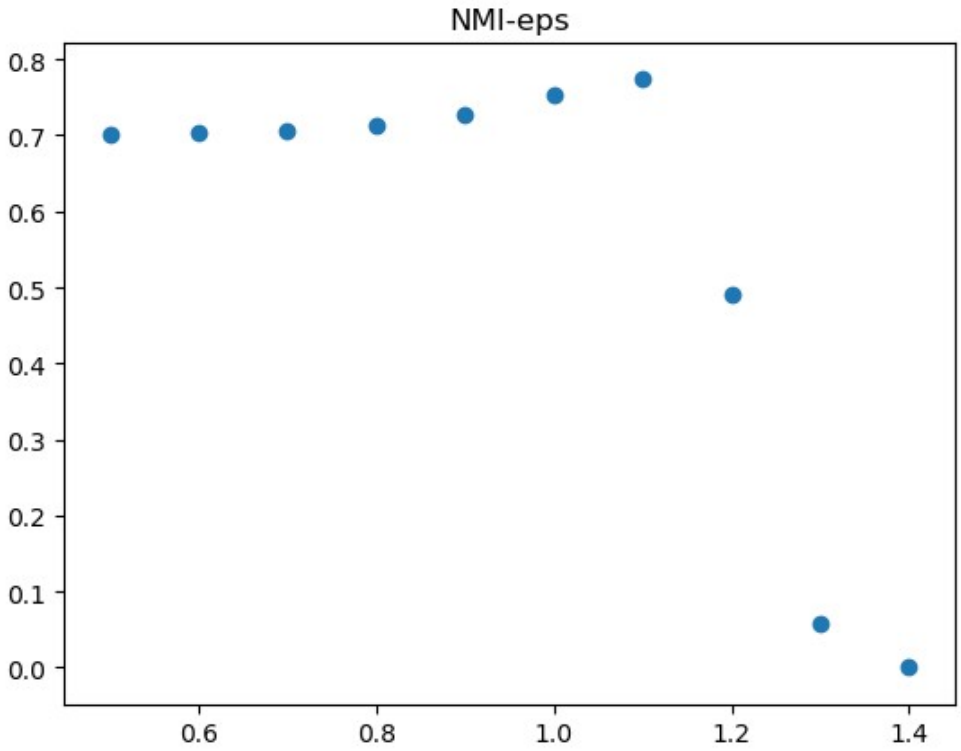
f) 重复 a),b),c),d),e)直至所有的点都被标记;
g) 将数据点分给被标记频率最高的簇,即完成分类;
3、 参数分析
a) bandwidth:浮点数,bandwidth 越小,分类越多;
(六)、DBSCAN
1、 算法原理
不同于划分和层次聚类方法,DBSCAN 将密度相连的点的最大集合定义为一个簇,即由密度可达关系导出的最大密度相连的样本集合就可以作为一个簇。
2、 算法步骤:
a) 检测数据库中尚未检查过的对象 p,如果 p 未被处理(归为某个簇或者标记为噪声),则检查其邻域,若包含的对象数不小于 minPts,建立新簇 C,将其中的所有点加入候选集 N;
b) 对候选集 N 中所有尚未被处理的对象 q,检查其邻域,若至少包含 minPts 个对象,则将这些对象加入 N;如果 q 未归入任何一个簇,则将 q 加入 C;
c) 重复步骤 b),继续检查 N 中未处理的对象,当前候选集 N 为空;
d) 重复步骤 a)~c),直到所有对象都归入了某个簇或标记为噪声。
3、 参数分析
a) Eps: 浮点数,表示两个互为邻居的 samples 之间的最大距离;
b) Min_samples:整数,sample 被视为 core point 的最小邻居数;
(七)、Gaussian_mixture1、算法原理
利用 E-M 算法,对高斯混合模型进行估计,计算出高斯混合模型中的参数,并利用该高斯混合模型对数据聚类结果进行估计。
2、参数分析
a) n_components: int 默认值为 1,表示高斯模型的个数
b) covariance_type: string 默认值为’full’,用于描述方差的类型,经过实验,当该参数设为’spherical’的时候,聚类效果最好。
五、聚类效果评估
1、k-means、spectral-clustering、avg- Agglomerative、ward- Agglomerative 四种聚类算法的 NMI 随着聚类个数的变化如下图所示:



2、 Mean-shift 聚类算法的聚类效果随着 bandwidth 的变化如下图所示:



3、 DBSCAN 聚类算法的聚类效果随着 eps 的变化情况如下图所示:4、Gaussian_mixture 模型的 NMI 随着 n_component 的变化如下图所示:



六、结论

本次实验总共实验了七种聚类算法,其中包括划分,层次等多种类型的聚类方式,有些聚类需要设置聚类个数,有些不需要;有些聚类的结果会受到初始值的影响,有些不会;有些聚类则需要设置其他的参数,否则甚至会导致结果完全错误。

通过这次实验,对这七种聚类算法有了大概的了解,熟悉了需要调节的参数的大概范围;了解了对聚类效果的评估手段 NMI。