中山大学计算机学院人工智能本科生实验报告

2022学年春季学期

课程名称：Artificial Intelligence

教学班级	人工智能（陈川）	专业（方向）	计算机科学与技术人工智能与大数据
学号	20337025	姓名	崔璨明

一、实验题目

在给定的文本数据集完成文本情感聚类训练，由于是无监督学习，所以只需要对训练集进行聚类

实验要求

1. 文本的特征可以使用TF-IDF（可以使用sklearn库提取特征）
2. 利用K-Means以及K-Means++完成对训练集的聚类，并计算准确率
3. 报告中需展示聚类中心以及各个样本降维后的分布结果
4. 需要提交简要报告+代码

二、实验内容

1、算法原理

K-Means算法原理

k-means算法又名k均值算法，K-means算法中的k表示的是将数据或样本聚类为k个簇，means代表取每一个聚类中数据值的均值作为该簇的中心，或者称为质心，即用每一个的类的质心对该簇进行描述。
其算法思想大致为：先从样本集中随机选取 k个样本作为簇中心，并计算所有样本与这k个“簇中心”的距离，对于每一个样本，将其划分到与其距离最近的“簇中心”所在的簇中，对于新的簇计算各个簇的新的“簇中心”。综上所述，实现kmeans算法的主要流程为：

1. 簇个数 k 的选择

2. 各个样本点到“簇中心”的距离
3. 根据新划分的簇，更新“簇中心”
4. 重复上述2、3过程，直至"簇中心"没有移动

K-Means++算法原理

k-means++算法是对k-means算法的改进，在k-means算法中，k个初始化的质心的位置选择对最后的聚类结果和运行时间都有很大的影响，因此需要选择合适的k个质心。如果仅仅是完全随机的选择，有可能导致算法收敛很慢。k-means++算法就是对k-means随机初始化质心的方法的优化。k-means++算法与k-means算法最本质的区别是在k个聚类中心的初始化过程。其主要流程为：

1. 在数据集中随机选择一个样本点作为第一个初始化的聚类中心
2. 计算样本中的每一个样本点与已经初始化的聚类中心之间的距离，并选择其中最短的距离，记为d_i
3. 选择一个新的数据点作为新的聚类中心，选择的原则是：距离较大的点，被选取作为聚类中心的概率较大
4. 重复2、3步，直到k个聚类中心都被确定
5. 对k个初始化的聚类中心，利用K-Means算法计算最终的聚类中心。

2、算法伪代码

k-means算法伪代码如下：

```
输入：待聚类的样本集dataset和聚类的数目k
输出：最终聚类结果cluster和每个聚类的质心centroids
def k_means(dataset,k):
        #随机取质心
        centroids=random_choose(dataset, k)
        #先进行首次聚类
    last_centroids,cluster=classify(dataset,centroids,k)
    while last_centroids!=new_centroids:
    #当质心还在变化时，继续进行计算和聚类
        newcentroids,cluster=classify(arr,newcentroids,k)
    #返回最后的结果
    return newcentroids,cluster

classify函数用于聚类
输入：样本集dataset，质心centroids，聚类数目k
输出：这一次的聚类结果cluster和每个聚类的质心newcentroids
def classify(dataset,centroids,k):
        distance[num of dataset][k] #用于保存距离
    for i in range(len(arr)):
        for j in range(len(centroids)):
            distance[i][j]=caculate(arr[i],centroids[j])
```

```
    minDistIndices = index of min of distance#选出最小距离的下标

        #聚类过程，每个样本分到相应的质心
    cluster=[[],[],[],[],[],[]]
    for i in range(len(minDistIndices)):
        cluster[minDistIndices[i]].append(arr[i])
    newCentroid=[]
    for i in range(k):
        newCentroid=average of cluster[i]#取每个聚类的平均值做为质心
    return newCentroid,cluster#返回结果

caculate函数用于计算距离
输入：两个样本向量a,b
输出：向量的欧式距离
def caculate(a,b):
        dis=0
    for i in range(len(a)):
        dis+=(a[i]-b[i])**2
    dis=dis**0.5
    return dis
```

k-means++算法的伪代码相比其k-means只是多了初始化的部分：

```
输入：待聚类的样本集dataset和聚类的数目k
输出：最终聚类结果cluster和每个聚类的质心centroids
def k_means_plus(dataset,k):
        #概率选取质心
        centroids=chose(dataset, k)
        #先进行首次聚类
    last_centroids,cluster=classify(dataset,centroids,k)
    while last_centroids!=new_centroids:
    #当质心还在变化时，继续进行计算和聚类
        newcentroids,cluster=classify(arr,newcentroids,k)
    #返回最后的结果
    return newcentroids,cluster

chose用于按照概率选择理想的初始质心
输入：数据集dataset，聚类数目k
输出：生成的质心centers
def chose(dataset,k):
    centers=[]
    #随机选取第一个点
    first_cent=random_choose(dataset)
    centers.append(first_cent)
    for cnt in range(k-1):
        distance=[]
        for i in range(len(arr)):
            dis=0
```

```
            for j in range(len(centers)):
                dis+=caculate(arr[i],centers[j])
            distance.append(dis)
        while 1:
                #将距离作为权重，距离越大选择的概率越大
            newone=random.choices from dataset,distance as weight
            #确保没有重复选择
            if newone not in centers:
                    break
        centers.append(newone)
    return centers
```

3、关键代码展示

在本次实验中，我利用tf-idf矩阵作为提取文本特征，在k-means算法计算向量相似度时采用欧式距离，依照上述伪代码编写程序，其关键代码和解析如下：

```
'''进行一次聚类的函数'''
def classify(arr,centroids,k):
    distance=[]#保存每个样本到各个质心距离的数组
    for i in range(len(arr)):
        distance.append([])
        for j in range(len(centroids)):
            distance[i].append(caculate(arr[i],centroids[j]))
    minDistIndices = np.argmin(distance, axis=1).tolist()#保存各个样本距离最小的质心
    cluster=[[],[],[],[],[],[]]#聚类列表
    for i in range(len(minDistIndices)):#进行聚类
        cluster[minDistIndices[i]].append(arr[i])
    newCentroid=[]
    for i in range(k):
        #调用numpy.mean()函数计算每个聚类的平均值，得到新质心
        newCentroid.append(np.mean(cluster[i],axis=0).tolist())
    return newCentroid,cluster

'''计算欧式距离'''
def caculate(a,b):
    dis=0
    for i in range(len(a)):
        dis+=(a[i]-b[i])**2
    dis**0.5
    return dis

'''k-means算法'''
def my_k_means(arr,k):
    # 随机取质心
    centroids = random.sample(arr, k)
```

```
    #centroids=chose(arr)
    cnt=1
    newcentroids,cluster=classify(arr,centroids,6)
    while cnt<20:
        cnt+=1
        newcentroids,cluster=classify(arr,newcentroids,6)
        #打印相关的信息，可以看到每个聚类中的数目
        for i in range(len(cluster)):
            print(len(cluster[i]))
        print("----------------------------------------------",cnt)
    return newcentroids,cluster
```

实现k-means++算法只需将k-means算法的随机初始选择质心更改为根据权重概率选择质心，而这个权重便是样本到已有质心的距离，即尽可能远地选择新的质心，其关键代码如下：

```
'''k-means++算法中选择初始质心的函数'''
def chose(arr):
    centers=[]
    first_cent=random.sample(arr,1)[0]#随机生成第一个质心
    centers.append(first_cent)
    for cnt in range(5):#选取剩下的
        distance=[]#保存每个样本到已选取的各个质心距离的数组
        for i in range(len(arr)):
            dis=0
            for j in range(len(centers)):
                dis+=caculate(arr[i],centers[j])#计算到已有的质心的距离
            distance.append(dis)
        while 1:#按概率选取
            newone=random.choices(arr,distance,k=1)[0]#距离作为权重矩阵
            if newone not in centers:#确保不重复
                break
        centers.append(newone)
    return centers
```

为进行可视化处理，还需要对聚类的结果进行降维，在本次实验中，我采取了两种降维的方法，第一种是调用TSNE进行降维，第二种是利用在上一次的实验中使用过的PCA模块进行降维，经过对比，采取了第一种降维方法，其关键代码如下：

```
'''对多维向量进行降维'''
def visual(X):
    tsne=manifold.TSNE(n_components=2,init='pca',random_state=501)
    X_tsne=tsne.fit_transform(X)
    x_min,x_max=X_tsne.min(0),X_tsne.max(0)
```

```
    X_norm=(X_tsne-x_min)/(x_max-x_min)
    return X_norm
```

除此之外，我们还需要对聚类结果进行评分，在这里我采取了两种方式来进行评估，先粗略计算聚类的正确率，然后计算兰德指数（Rand index）来评估聚类的性能，兰德指数需要给定实际类别信息C,假设K是聚类结果，a表示在C与K中都是同类别的元素对数，b表示在C与K中都是不同类别的元素对数。评价同一object在两种分类结果中是否被分到同一类别。其公式如下：

数学公式：

如果C是真实类别，K是聚类结果，我们定义a和b分别是：

a: 在C和K中都是同一类别的样本对数

b: 在C和K中都是不同类别的样本对数

raw Rand Index 的公式如下:

$$\mathrm{RI} = \frac{a+b}{C_2^{n_{samples}}}$$

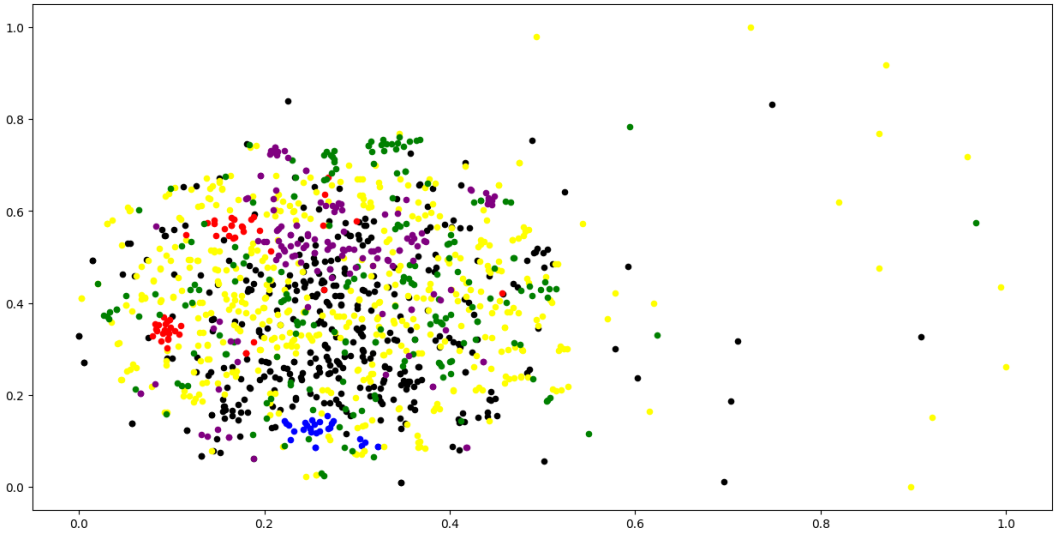
一般来讲，兰德指数越接近1则表面聚类结果越好。
实现代码如下：

```
ra=0
for i in range(len(Cluster)):
        for j in range(len(Cluster[i])):
                for k in range(j+1,len(Cluster[i])):
                        if types[arr.index(Cluster[i][j])] ==
types[arr.index(Cluster[i][k])]:
                        ra+=1

rb=0
for i1 in range(len(Cluster)):
    for i2 in range(i1+1,len(Cluster)):
        for j1 in range(len(Cluster[i1])):
            for j2 in range(len(Cluster[i2])):
                if types[arr.index(Cluster[i1][j1])] !=
types[arr.index(Cluster[i2][j2])]:
                rb+=1
#计算组合数
com=math.factorial(len(arr))/(2*math.factorial(len(arr)-2))
#计算兰德指数
rand_index=(ra+rb)/com
print("rand index = ",rand_index)
```

三、实验结果

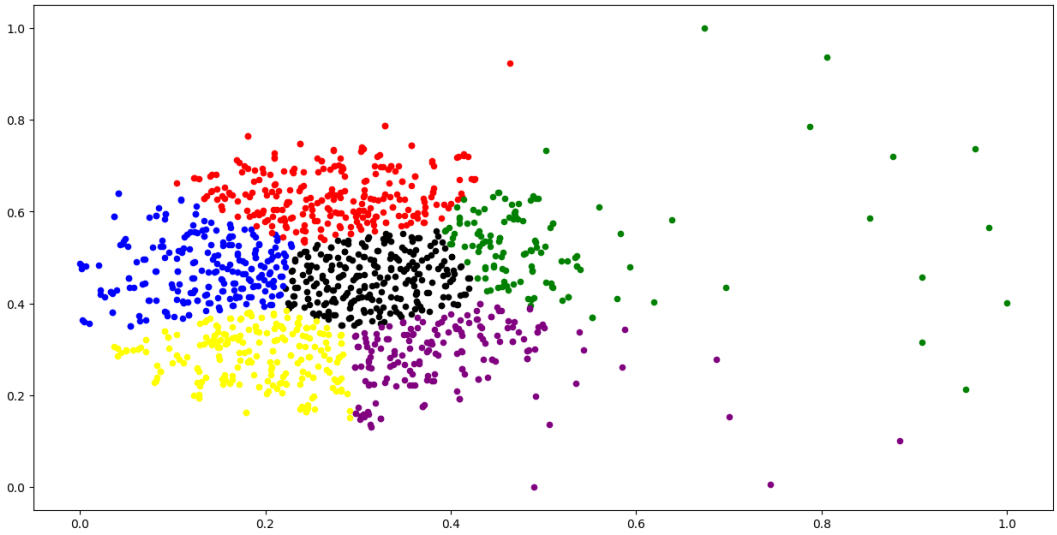
将所有文本数据作为一个数据集进行聚类，用tf-idf矩阵提取特征，并增加停用词处理，首先利用k-means算法进行聚类，并将聚类结果进行降维，采用python中的`plt`库进行可视化输出，聚类结果如下：



可能是降维的问题，将许多样本混杂在了一起……粗略计算聚类的准确率和兰德指数如下：

```
rate: 0.3893805309734513
rate: 0.36065573770491804
rate: 0.4087403598971722
rate: 0.3181818181818182
rate: 0.3333333333333333
rate: 0.3229166666666667
rand index =  0.6622606180348404
PS E:\VSCODE\py>
```

采用k-means++算法进行聚类，同样将聚类结果进行降维，采用python中的`plt`库进行可视化输出，在进行了多次实验后，产生最好的聚类结果如下：



粗略计算聚类的准确率和兰德指数如下

```
---------------------------------------- 20
rate: 0.36492890995260663
rate: 0.39271255060728744
rate: 0.3932038834951456
rate: 0.31277533039647576
rate: 0.44715447154471544
rate: 0.2727272727272727
rand index =  0.6783765286225287
PS E:\VSCODE\py>
```

兰德指数为0.67，取得了不错的聚类效果。

四、参考资料

- [哈工大停用词表](GitHub - goto456/stopwords: 中文常用停用词表（哈工大停用词表、百度停用词表等）)
- (113条消息) 机器学习的评价指标-Rand index_ddupyy的博客-CSDN博客_rand指数