**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 | **系统结构班** | 专业（方向） | **计算机科学与技术** |
| 学号 | **21307358** | 姓名 | **曾慧蕾** |

# 实验题目

* 在给定文本数据集完成文本情感聚类训练
* **要求**

1. **文本的特征可以使用TF或TF-IDF （可以使用sklearn库提取特征）**
2. **利用K-means，K-means++对文本特征进行聚类**
3. **计算calinski\_harabasz\_score（越高越好）**
4. **可调用sklearn.metrics.calinski\_harabasz\_score函数计算**
5. **需提交实验报告+代码**
6. **实验报告应包含样本的可视化展示，可以考虑利用TSNE将文本特征投影到直角坐标系中进行展示（可以调用sklearn库的TSNE投影，可视化工具不限）**

# 实验内容

1. 算法原理

**K-means：**

k-means算法是一种基于距离的聚类算法，它的目标是将数据集划分为k个不同的簇，使得每个数据点都属于离它最近的簇。k-means算法的基本思路是：首先随机选择k个初始聚类中心，然后将每个数据点分配到离它最近的聚类中心所在的簇中，接着重新计算每个簇的聚类中心，重复上述过程，直到聚类中心不再发生变化或达到最大迭代次数为止。

**基本思想：**

1. 将数据集划分成K个不相交的块
2. 每个块都由一个中心点(centroid)联系起来
3. 数据集中每个点都被划分到唯一的最接近的中心点所在的块中

**实现细节：**

1. 首先随机初始化K个中心点
2. 对于数据集中的N个点，计算其到这K个中心点的距离(度量使用SSE)
3. 样本点被划分到距离最近的中心点所在的块
4. 数据集中所有样本均划分好块后，对中心点进行更新(一般更新为块中样本点的均值)
5. 判断是否收敛
6. 循环，直到质心不变化或者限定循环一定次数

**K-means++：**

k-means++算法是k-means算法的一种改进版本，它通过改进初始聚类中心的选择方法，可以更好地避免聚类结果受初始聚类中心的影响，使初始聚类中心的分布更加均匀，避免了初始聚类中心过于集中或过于分散的情况，从而提高了聚类结果的质量。

**基本思想：**

1. 随机选择一个数据点作为第一个聚类中心
2. 对于每个数据点，计算它与已有聚类中心的距离的平方，选择距离最近的一个数据点作为下一个聚类中心
3. 重复上述过程，直到选择k个聚类中心为止

**实现细节：**

1. 首先随机选择一个数据集中的点作为初始化的聚类中心
2. 对于数据集中的其他样本点𝑥𝑥𝑖𝑖，计算其与聚类中心的距离，并记录最短距离𝑑𝑖
3. 按概率采样一个样本点，距离越大，被采样的概率也就越大
4. 重复上述过程，直到k个聚类中心都被确定
5. 利用K-means进行聚类
6. 伪代码

Kmeans：

输入：数据集

输出：数据集划分后的质心以及各样例所属簇

def kmeans(dataset, k, max\_iter, tol):

    # 随机选择k个数据点作为初始聚类中心

    centroids = dataset[np.random.choice(len(dataset), k, replace=False)]

    cluster = np.zeros(len(dataset))

    for i in range(max\_iter):

        # 将每个数据点分配到离它最近的聚类中心所在的簇中

        for i in range(len(dataset)):

            distances = norm(dataset[i] - centroids, axis=1)

            cluster[i] = argmin(distances)

        # 重新计算每个簇的聚类中心

        new\_centroids = np.zeros\_like(centroids)

        for i in range(k):

            new\_centroids[i] = np.mean(dataset[cluster == i], axis=0)

        # 计算聚类中心的移动距离

        move\_distance = norm(new\_centroids - centroids)

        # 更新聚类中心

        centroids = new\_centroids

        # 如果聚类中心的移动距离小于阈值，退出迭代

        if move\_distance < tol:

            break

    # 将每个数据点分配到最终的聚类中心所在的簇中

    for i in range(len(dataset)):

        distances = norm(dataset[i] - centroids, axis=1)

        cluster[i] = np.argmin(distances)

    return cluster, centroids

k-means++：

k-means++算法只是改进了初始聚类中心的选择方法，而聚类的过程仍然是使用k-means算法进行的。

输入：数据集，簇数

输出：初始质心

def kmeans\_plus\_plus(dataset, k):

    # 从数据集中随机选择一个数据点作为第一个聚类中心

    centroids = [dataset[random.randint(len(dataset))]]

    # 选择剩余的k-1个聚类中心

    for i in range(k - 1):

        # 对于每个数据点，计算它与已有聚类中心的距离的平方

        distances = np.array([min([np.linalg.norm(x - c) for c in centroids]) for x in dataset])

        # 选择距离最近的一个数据点作为下一个聚类中心

        probabilities = distances / distances.sum()

        next\_centroid = dataset[np.random.choice(len(dataset), p=probabilities)]

        centroids.append(next\_centroid)

    # 使用k-means算法进行聚类

    kmeans = KMeans(n\_clusters=k, init=np.array(centroids))

    kmeans.fit(dataset)

    # 输出聚类结果

    return labels\_, cluster\_centers\_

1. 关键代码展示（带注释）

首先对文本进行特征提取

    #构建向量空间模型 提取特征词

    tv = TfidfVectorizer(use\_idf=False)

    tv\_fit = tv.fit\_transform(example) #频率

    ft\_name = tv.get\_feature\_names\_out() #提取的特征词

    arr = tv\_fit.toarray().tolist()  #矩阵转列表

然后开始对数据集进行分类

def k\_means(arr,k):

    dot\_set=k\_means\_plus(arr,k)

    new\_dot\_set=[]

    for i in range(0,800):

        label,classify,new\_dot\_set=judgement(arr,k,dot\_set)

        dot\_set=new\_dot\_set

    return label,classify,dot\_set

其中涉及到的k\_means\_plus是k-means++算法，在初始数据筛选最合适的k个质心

def k\_means\_plus(arr,k): #k-means++ 让初始聚类中心之间的相互距离尽可能远

    first\_dot=random.randint(0,len(arr)-1)

    ans=[arr[first\_dot]]

    while k-1:

        dis=[] #每次选取都更新数组

        for i in range(0,len(arr)):

            d=0

            for j in range(0,len(ans)):

                d+=distance(ans[j],arr[i]) #根据答案集的大小更改循环次数

            dis.append(d)

        while True:#按概率采样一个样本点，距离越大，被采样的概率也就越大

            dot=random.choices(arr,dis,k=1) #轮盘赌

            if dot not in ans:

                ans.append(dot[0])

                break

        k-=1

    return ans

随后可以采取两种措施：循环一定次数或者循环直到质心不变，我这里选择了前者。每次循环都要调整质心，以及根据新的质心对数据集重新分类

def judgement(arr,k,dot): #每一次根据原来的质心计算分类 并算新的质心

    classify=[[],[],[],[],[],[]]

    label=[]

    new\_dot\_set=[]

    d=[0,0,0,0,0,0]

    #通过距离分类

    for i in range(0,len(arr)):

        d=[0,0,0,0,0,0]

        for j in range(0,k):

            d[j]=distance(arr[i],dot[j]) #计算并存入与每个质心的距离

        temp=d.index(min(d))

        classify[temp].append(arr[i]) #聚类

        label.append(temp)

    #计算新的质心

    for i in range(0,k):

        t=get\_centroid(classify[i])

        new\_dot\_set.append(t)

    return label,classify,new\_dot\_set

其中涉及到的质心计算函数为：

def get\_centroid(arr): #算质心

    aver=[0 for i in range(0,len(arr[0]))]

    for i in range(0,len(arr)+1):

        if i==len(arr):

            for j in range(0,len(arr[0])):

                aver[j]/=len(arr) #得到平均值点

        else:

            for j in range(0,len(arr[0])):

                aver[j]+=arr[i][j]

    min=9999999

    ans=arr[0]

    for i in range(0,len(arr)):

        d=distance(aver,arr[i])

        if d<min:

            min=d

            ans=arr[i]

    return ans

1. 创新点&优化（如果有）

为了更方便可视化，我在最后采取了降维的方法。关于这一点有两种思路：可以先降维后分类或者先分类后降维。前者可视化之后分类效果更明显，后者是在高维层次上的划分所以在降维之后在二维平面上看起来会杂乱一点。降维代码如下所示：

def dimension\_reduction(X\_):

    tsne2d=TSNE(n\_components=2,perplexity=1,init='pca',random\_state=0)

    X=np.array(X\_)

    X\_tsne\_2d=tsne2d.fit\_transform(X)

    x\_min, x\_max=np.min(X\_tsne\_2d,0),np.max(X\_tsne\_2d,0)

    ans=(X\_tsne\_2d-x\_min)/(x\_max-x\_min)

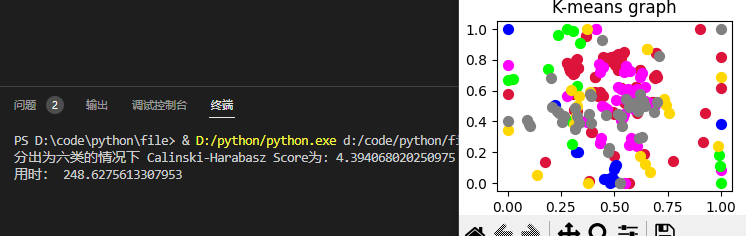
    return ans

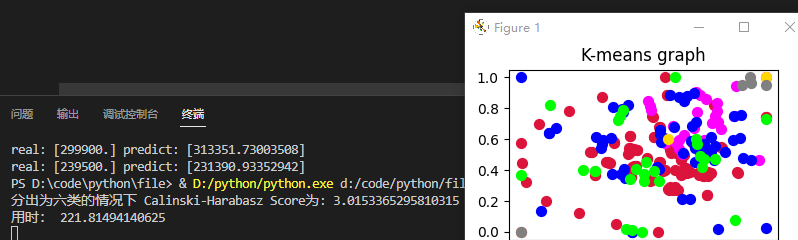
# 实验结果及分析

1. 实验结果展示示例与评测指标展示及分析

本实验使用k-means++算法优化，分为先分类后降维和先降维后分类两种顺序，这里先展示先降维后分类的结果

**先降维后分类：**





**这个操作顺序的分类是在高维层次上进行的，因此先分类再将这些点簇投影到同一个平面时，结果往往会显得杂乱无章，计算出来的calinski\_harabasz\_score也会偏低。上图就是一个例子。**

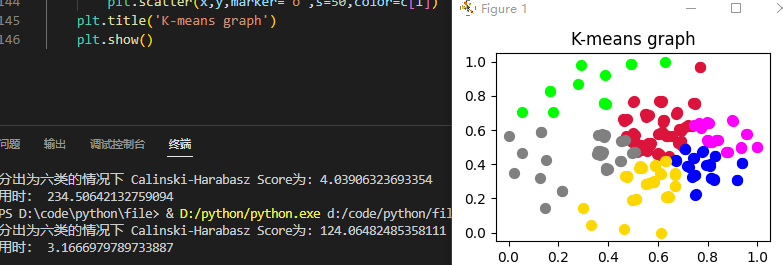
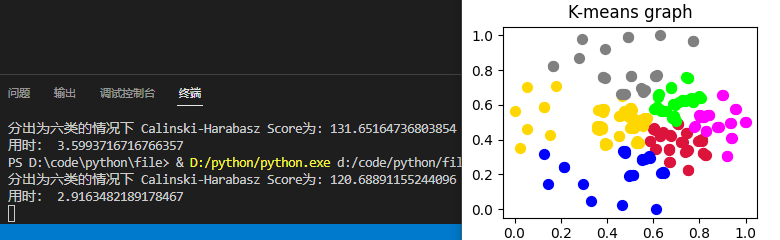
**在测试多次后，得到的结果如下：**

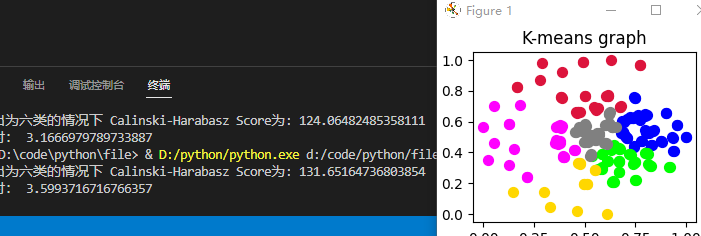
|  |  |  |
| --- | --- | --- |
| **序号** | **calinski\_harabasz\_score** | **用时** |
| **1、** | **3.38202656** | **206.406309** |
| **2、** | **3.01533652** | **221.814941** |
| **3、** | **4.03906323** | **234.506421** |
| **4、** | **3.21689291** | **236.431714** |
| **5、** | **3.00654821** | **220.614866** |
| **6、** | **4.39406802** | **248.627561** |
| **7、** | **3.29427915** | **212.672605** |

**其中，calinski\_harabasz\_score最高时为：4.39406802**

**先分类后降维：**

**这个顺序是先将点降维在同一个平面上再进行分类，所以结果会很明显。**





**在测试多次后，得到的结果如下：**

|  |  |  |
| --- | --- | --- |
| **序号** | **calinski\_harabasz\_score** | **用时** |
| **1、** | **124.06482485358111** | **3.1666979789733887** |
| **2、** | **131.65164736803854** | **3.5993716716766357** |
| **3、** | **120.68891155244096** | **2.9163482189178467** |
| **4、** | **124.94968788000804** | **2.6244406700134277** |
| **5、** | **140.0745139191649** | **2.4495058059692383** |
| **6、** | **122.98139648315401** | **3.1035890579223633** |
| **7、** | **124.62445430053809** | **3.5551695823669434** |

**其中，calinski\_harabasz\_score最高时为：140.0745139191649**

**可见，先降维再分类时，在需要的维度上，算法效率将非常高（比如此次实验所需维度为2）**

# 参考资料

[基于sklearn的聚类算法的聚类效果指标\_sklearn 聚类评价指标](https://blog.csdn.net/qq_27825451/article/details/94436488?utm_medium=distribute.pc_aggpage_search_result.none-task-blog-2~aggregatepage~first_rank_ecpm_v1~rank_v31_ecpm-1-94436488.pc_agg_new_rank&utm_term=DBI%E6%8C%87%E6%A0%87%20sklearn&spm=1000.2123.3001.4430)

[TSNE实现降维及可视化\_tsne降维](https://blog.csdn.net/qq_41076797/article/details/115508184)

**超算习堂-课时24-第12讲**

**超算习堂-课时20-第12讲实验课**