计算机学院 社交媒体与舆情分析 课程实验报告

实验题目： 3 Clustering Classification		**学号：** 201800130112
日期： 2021.10.20	**班级：** 计科 18.3	**姓名：** 赵子涵
Email： zih_an@163.com		

实验方法介绍：

一、K-Means

典型的聚类算法，算法步骤：

1. 设置类别数 class_num
2. 随机初始化分配 class_num 个中心位置（centroid）
3. 为每个数据分配最近的中心位置（cluster）
4. 计算当前每个类的新中心位置（centroid）
5. 重复第 2,3,4 步，除非遇到终止条件：中心位置基本无变化，或循环次数达到上限。

二、BoW（Bag of Words）词袋模型

词袋模型是指，将所有文本所有词形成一个词的集合，对于每一条文本，文本中有的词在对应位置标记 1，否则标记 0。

实验过程描述：（不要求罗列完整源代码）

一、导入数据并分词

数据说明：

`Twitter_data`: This file contains 29846 data,and each of them has 8 items

- "userName":用户名
- "clusterNo":类别
- "text":Twitter内容
- "timeStr":时间戳
- "tweeId":用户Id
- "errorCode":状态码
- "textCleaned":去除链接等特殊符号只保留文本的处理
- "relevance":

本实验选择使用 textCleaned 作为聚类的特征维度（即用户发送的 twitter 文本），将文本直接按空格分词并使用词袋模型 BoW 向量化之后，使用用 KMeans 算法聚类。将聚类后的维度，与直接按 clusterNo 维度聚类的结果进行对比，得到 KMeans 聚类的正确率。

- 导入数据：

```python
import json
# 将数据读取成dict格式便于后续的操作
Twitter_data=[]
with open("Twitter_data")as f:
    for line in f:
        # print(line)
        Twitter_data.append(json.loads(line))
```

- 分词：

```python
token_textCleaned = []
words = set([])
for item in Twitter_data:
    tokens = item["textCleaned"].split(" ")
    token_textCleaned.append(tokens)
    for token in tokens:
        words.add(token)

num_words_max = len(words)
```

- 词袋模型：

```python
import numpy as np

bow_dict = dict()
for i, word in enumerate(words):
    bow_dict[word] = i

vec_textCleaned = np.zeros((len(token_textCleaned), num_words_max))
for i, sentence in enumerate(token_textCleaned):
    for word in sentence:
        j = bow_dict[word]
        vec_textCleaned[i][j] = 1
```

二、KMeans 聚类

在本实验中，根据实验指导书，设置 200 个类别。

```python
class KMeans():
    def __init__(self, data, num_classes, max_iter=200):
        self.num_classes = num_classes
        self.src_data = data
        self.max_iter = max_iter
        self.m_examples, self.n_features = data.shape

        self.label = np.zeros(self.m_examples)
        self.clusters = [[] for i in range(num_classes)]  ## idx_list of each class in src_data
        ## center vectors
        init_cen_idx = np.random.choice(self.m_examples, num_classes, replace=False)  ## init randomly first
        self.centroid = self.src_data[init_cen_idx]

    def run(self, threshold=1e-2):
        for _ in range(self.max_iter):
            print("cluster")
            self.clusters = [[] for i in range(self.num_classes)]
            self._cluster(self.centroid)
            print("centroid")
            newCentroid = self._genCentroid(self.clusters)
            if self._edis(self.centroid, newCentroid) < threshold:
                print("bbbbbreak")
                break
            self.centroid = newCentroid
        return self.label

    def _cluster(self, centroid):
        for idx, sample in enumerate(self.src_data):
            lbl, dis = -1, float("inf")
            for cls in range(self.num_classes):
                tmp = np.sum((sample - centroid[cls])**2)
                if tmp < dis:
                    lbl = cls
                    dis = tmp
            self.label[idx] = lbl  # record the class for this sample
            self.clusters[lbl].append(idx)  # add this sample to the class

    def _genCentroid(self, clusters):
        newCentroid = np.zeros((self.num_classes, self.n_features))
        for i, cluster in enumerate(clusters):
            cluster_mean = np.mean(self.src_data[cluster], axis=0)
            newCentroid[i] = cluster_mean
        return newCentroid

    def _edis(self, cen1, cen2):
        return np.sum(np.sqrt(np.sum((cen1-cen2)**2, axis=1)))
```

```python
# km = KMeans(np.array([[0, 1, 1], [1, 0, 0], [1, 1, 0], [1, 0, 1], [0, 0, 1], [1, 1, 1]]), 3, 10)
# lbl = km.run()
# print(lbl)
km = KMeans(vec_textCleaned, 200, 3)
lbl = km.run()
```

三、结果测试

思想：遍历聚类后的每一组，取每一组数据的 clusterNo 维度中，出现最频繁的作为该组的类别。依次统计每组的正确率，取平均值得到该 KMeans 聚类结果的正确率情况，截图如下：

```python
def evaluation(data_dict, clusters):
    evl = np.zeros(len(clusters))
    for idx, clstr in enumerate(clusters):
        clsOrg = [Twitter_data[i]["clusterNo"] for i in clstr]
        if len(clsOrg) > 0:
            mainCls = max(clsOrg, key=clsOrg.count)  # 统计出现最多次的元素
            evl[idx] = clsOrg.count(mainCls) / len(clsOrg)
        else:
            evl[idx] = 0
    return evl
```

```python
evl = evaluation(Twitter_data, km.clusters)
print(np.mean(evl))
```

```
0.9471817805273113
```

结论分析：

使用 KMeans 对 twitter 文本聚类与所属类别关系基本一致，正确率约为 94.72%。在实验过程中，发现聚类运行速度比较慢，主要是数据维度太高、量太大导致的。

根据该结果，发现实际文本中 clusterNo 最大为 225，但是初始选择的聚类个数为 200 个，所以必定导致正确率偏低，但是发现正确率仍然较高，也可以验证算法正确。

对于本实验中使用的词袋模型 BoW，下一步可考虑使用 TFIDF 测试。

结论：

使用 BoW 向量化文本，并用 KMeans 对文本聚类的效果不错，可以很有效的完成。但是，KMeans 初始的类别选择与最终聚类的结果好坏息息相关。

核心代码——词袋模型

```python
import numpy as np

bow_dict = dict()
for i, word in enumerate(words):
    bow_dict[word] = i

vec_textCleaned = np.zeros((len(token_textCleaned), num_words_max))
for i, sentence in enumerate(token_textCleaned):
    for word in sentence:
        j = bow_dict[word]
        vec_textCleaned[i][j] = 1
```

KMeans 聚类

```
class KMeans():
    def __init__(self, data, num_classes, max_iter=200):
        self.num_classes = num_classes
        self.src_data = data
        self.max_iter = max_iter
        self.m_examples, self.n_features = data.shape

        self.label = np.zeros(self.m_examples)
        self.clusters = [[] for i in range(num_classes)]  ## idx_list of each class in src_data
        ## center vectors
        init_cen_idx = np.random.choice(self.m_examples, num_classes, replace=False)  ## init randomly first
        self.centroid = self.src_data[init_cen_idx]

    def run(self, threshold=1e-2):
        for _ in range(self.max_iter):
            print("cluster")
            self.clusters = [[] for i in range(self.num_classes)]
            self._cluster(self.centroid)
            print("centroid")
            newCentroid = self._genCentroid(self.clusters)
            if self._edis(self.centroid, newCentroid) < threshold:
                print("bbbbbreak")
                break
            self.centroid = newCentroid
        return self.label

    def _cluster(self, centroid):
        for idx, sample in enumerate(self.src_data):
            lbl, dis = -1, float("inf")
            for cls in range(self.num_classes):
                tmp = np.sum((sample - centroid[cls])**2)
                if tmp < dis:
                    lbl = cls
                    dis = tmp
            self.label[idx] = lbl  # record the class for this sample
            self.clusters[lbl].append(idx)  # add this sample to the class

    def _genCentroid(self, clusters):
        newCentroid = np.zeros((self.num_classes, self.n_features))
```

```
        for i, cluster in enumerate(clusters):
            cluster_mean = np.mean(self.src_data[cluster], axis=0)
            newCentroid[i] = cluster_mean
        return newCentroid

    def _edis(self, cen1, cen2):
        return np.sum(np.sqrt(np.sum((cen1-cen2)**2, axis=1)))
```