信息处理技术 大作业

From: 梁鑫宇 3160104494

1. 题目:

综合运用分词，聚类等这学期学习过的内容；对校综合服务网上的通知进行聚类分析

2. 数据

使用的是 2017-12-07 至 2018-07-04 的通知数据，共 1403 条。

3. 实现

同之前分词作业一样，利用 json 文件保存中间分词结果，再利用 K-means 方法聚类。其中仍然使用 SSE 控制循环结束。

通知聚类与前面数字聚类比较不同的两点

1) 两点间距离，这里选用两个标题不重合的分词数量作为距离；
2) 类中心点，文字内容较难以坐标形式定位，此问题属实困扰。后来想到一种解决方案是通过找出现有类的最高频词汇，生成一个虚拟的高频标题作为中心点。

4. 改进

完成程序后可以跑出较为满意的结果，但是可以发现一些常用词的影响，思考可能是由于类中心点的取法加重了这种信息量少的高频词的影响，思考如果设置停用词应该会有利于聚类结果，但由于时间关系没有很好地完成。QAQ

源代码

load.py

```
import json

def MaxSize(json_name):

    MaxSize = 0

    with open(json_name) as file_obj:

        words = json.load(file_obj)

    for word in words:

        if len(word) > MaxSize:

            MaxSize = len(word)

    return MaxSize



def GetList(json_name):

    with open(json_name) as file_obj:

        words = json.load(file_obj)

    return words
```

search.py

```
def check(word,words,left,right):

    while left<=right:

        mid = int((left+right)/2)

        if word == words[mid]:

            return True
```

```
        elif word < words[mid]:
            right = mid - 1
        else:
            left = mid + 1
    if left > right:
        return False
    else:
        return True
```

CSVtoJSON.py

```
import csv
import load
import json
from search import check

json_name = "words_list.json"
MaxSize = load.MaxSize(json_name)
words = load.GetList(json_name)

def GetWord(UserStr,MaxSize,words):
    list_size = len(words)
    temp_word = UserStr[0:MaxSize]
    while len(temp_word) > 1:
        if check(temp_word,words,0,list_size-1):
            return temp_word
        else:
            temp_word = temp_word[0:len(temp_word)-1]
    return temp_word


def divide(fileName):
    with open(fileName,"r") as fileobj:
        csvReader = csv.reader(fileobj)
        headers = next(csvReader)
        dividedNotice = []
        NoticeDic = {}
```

```
        count = 0
        for row in csvReader:
            notice = row[0]
            # for i in range(0,len(notice)):
            #     print(notice[i])
            #     if notice[i] < u'\u4e00' or notice[i] > u'\u9fa5':
            #         print(notice[i])
            #         notice = notice[0:i]+notice[i+1:]
            notice = notice[0:-11]
            NoticeDic[count] = notice
            count += 1
            dividedwords = []
            while(len(notice)):
                if len(notice) >= MaxSize:
                    word = GetWord(notice, MaxSize, words)
                    dividedwords.append(word)
                    notice = notice[len(word):]
                else:
                    word = GetWord(notice, len(notice), words)
                    dividedwords.append(word)
                    notice = notice[len(word):]
            dividedNotice.append(dividedwords)
        with open("dividedNotice.json","w") as store:
            json.dump(dividedNotice, store)
        with open("NoticeDic.json", "w") as store:
            json.dump(NoticeDic, store)

divide("zjuNotice.csv")
```

classpoint.py

```
class point(object):
    def __init__(self,words):
        self.words = words
        self.inclause = -1

    def distance(self, other):
        return len([i for i in self.words if i not in other.words])

    def __nearest(self,dists):
        min = float('Inf')
        for dist in dists:
            if dist < min:
                min = dist
        return dists.index(min)
```

```
    def group(self, centroids):
        dists = []
        for centroid in centroids:
            dists.append(self.distance(centroid))
        self.inclause = self.__nearest(dists)

    def show(self):
        noticeString = ""
        for word in self.words:
            noticeString += word
        print(noticeString)
```

functions.py

```
import random
import operator
from classpoint import point

def avg(points):
    countdic = {}
    centerpoint = []
    if len(points):
        for apoint in points:
            for word in apoint.words:
                if word in countdic:
                    countdic[word] += 1
                else:
                    countdic[word] = 1
        sortedWord = sorted(countdic.items(), key=operator.itemgetter(1),
reverse=True)
        for word in sortedWord:
            centerpoint.append(word[0])
        if len(centerpoint) > 6:
            return point(centerpoint[0:6])
        else:
            return point(centerpoint)

def FreshClauses(points, Clauses):      #Clauses is a 2-dimension, k-length list,
storing point[]
    for Clause in Clauses:
        Clause.clear()
    for point in points:
        Clauses[point.inclause].append(point)
```

```
def SSE(Clauses,centroids):
    sse = 0
    for Clause in Clauses:
        for point in Clause:
            sse += pow(point.distance(centroids[point.inclause]),2)
    return sse

def TryNextLoop(points,centroids,Clauses):
    for Clause in Clauses:
        centroids[Clauses.index(Clause)] = avg(Clause)
    for point in points:
        point.group(centroids)
    FreshClauses(points, Clauses)
    return SSE(Clauses,centroids)



def adjust(points, centroids, Clauses):      #centroids is a list of centroid, with
a size of k
    sse = SSE(Clauses,centroids)
    while 1:
        temp_points = points        #back up
        temp_centroids = centroids
        temp_Clauses = Clauses
        new_sse = TryNextLoop(temp_points,temp_centroids,temp_Clauses)
        if new_sse < sse:
            points = temp_points    #fresh
            centroids = temp_centroids
            Clauses = temp_Clauses
            sse = new_sse
        else:
            break

def GenerateCentroids(points,K,Clauses):
    GenerateSet = []
    while len(set(GenerateSet)) < K:
        GenerateSet.append(random.randint(0,len(points)))
    Centroids = []
    for i in GenerateSet:
        Centroids.append(points[i])
    for point in points:
        point.group(Centroids)
    FreshClauses(points, Clauses)
    return Centroids
```

main.py

```
from classpoint import point
import functions
import json
import load

K = input("Please input how many centroid you'd like to have:   ")
rawpoints = load.GetList("dividedNotice.json")
Clauses = []
points = []
for rawpoint in rawpoints:
    points.append(point(rawpoint))

for i in range(0,int(K)):
    Clauses.append([])

if int(K)>len(points):
    exit("Too Many centroids!")

Centroids = functions.GenerateCentroids(points,int(K),Clauses)
functions.adjust(points,Centroids,Clauses)
print("------")
for Clause in Clauses:
    if(len(Clause)):
        print(functions.avg(Clause).words)
    for point in Clause:
        point.show()
    print("------")
```