20377135 week2

20377135 邹诺天来

1.使用 danmuku.csv，其中一个弹幕可以视为一个文档（document），读入文档并分词（可以使用 jieba 或 pyltp）

2.过滤停用词（可用 stopwords_list.txt，或自己进一步扩充）并统计词频，输出特定数目的高频词和低频词进行观察。建议将停用词提前加入到 jieba 等分词工具的自定义词典中，避免停用词未被正确分词

文档读入：

```Python
inputs = open("C:\\Users\\dexter\\Desktop\\danmuku.csv",encoding='UTF-8') #
outputs = open('C:\\Users\\dexter\\Desktop\\output.txt', 'w',encoding='UTF-
reader=csv.reader(inputs)
header_row=next(reader)
content = []
for row in reader:
    content.append(row[0])
```

停用词读入函数：

```Python
# 创建停用词list
def stopwordslist(filepath):
    stopwords = [line.strip() for line in open(filepath, 'r',encoding='UTF-
    return stopwords
```

过滤停用词 + 分词函数：

```Python
# 对句子进行分词
def seg_sentence(sentence):
    sentence_seged = jieba.cut(sentence.strip())
    stopwords = stopwordslist('C:\\Users\\dexter\\Desktop\\stopwords_list.t
    outstr = ''
    for word in sentence_seged:
```

```
        if word not in stopwords:
            if word != '\t':
                outstr += word
                outstr += " "
    return outstr
```

词频统计：

```Python
# WordCount
with open('C:\\Users\\dexter\\Desktop\\output.txt', 'r',encoding='UTF-8') a
    data = jieba.cut(fr.read())
data = dict(Counter(data))
```

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\dexter\AppData\Local\Temp\jieba.cache
Loading model cost 0.557 seconds.
Prefix dict has been built successfully.
```

程序运行

3.根据词频进行特征词筛选，如只保留高频词，删除低频词（出现次数少于 5 之类），并得到特征词组成的特征集。

删除低频词 + 得到特征集：

```Python
high = []
with open('C:\\Users\\dexter\\Desktop\\cipin.txt', 'w',encoding='UTF-8') as
    for k, v in data.items():
        fw.write('%s,%d\n' % (k, v))
        if v>=5:
            high.append(k)
```

```
[' ', '\n', '小时', '前', '点', '刚刚', '辽', '两', '囍', '发现', '喜欢', '武汉',
```

部分高频词特征集

4.利用特征集为每一条弹幕生成向量表示，可以是 0，1 表示（one-hot，即该特征词在弹幕中是否出现）也可以是出现次数的表示（该特征词在弹幕中出现了多少次）。注意，可能出现一些过短的弹幕，建议直接过滤掉。

每一条弹幕的特征向量表示：

Python

```python
n=len(high)
sum=0
one_hot=[[0 for i in range(len(high))] for i in range(len(content))]
for i in range(len(content)):
    sum+=len(content[i].split())
    for j in range(len(high)):
        if high[j] in content[i]:
            one_hot[i][j]+=1
```

注：此处虽然取名为 onehot，但是是以特征词出现次数进行计数来实现弹幕的特征向量，下文中 onehot 同理

5.利用该向量表示，随机找几条弹幕，计算不同弹幕间的语义相似度，可尝试多种方式，如欧几里得距离或者余弦相似度等，并观察距离小的样本对和距离大的样本对是否在语义上确实存在明显的差别。请思考，这种方法有无可能帮助我们找到最有代表性的弹幕？

基于欧几里得距离计算随机 10 条弹幕语义相似度函数：

Python

```python
def onehot(content,high,one_hot):
    list = range(len(content))
    num = random.sample(list, 10)
    dis=[[0 for i in range(10)]for i in range(10)]
    ans=0
    for i in range(10):
        for j in range(10):
            for k in range(len(high)):
                ans+=(one_hot[i][k]-one_hot[j][k])*(one_hot[i][k]-one_hot[j
            dis[i][j]=dis[i][j]=math.sqrt(ans)
            ans=0
    for i in range(10):
        print(dis[i])
```

```
[0.0, 1.7320508075688772, 1.4142135623730951, 1.4142135623730951, 2.6457513110645907, 1.7320508075688772, 1.4142135623730951, 1.4142135623730951, 2.0, 1.4142135623730951
[1.7320508075688772, 0.0, 1.7320508075688772, 1.7320508075688772, 2.0, 0.0, 1.7320508075688772, 1.7320508075688772, 1.0, 1.7320508075688772]
[1.4142135623730951, 1.7320508075688772, 0.0, 1.4142135623730951, 2.6457513110645907, 1.7320508075688772, 1.4142135623730951, 1.4142135623730951, 2.0, 1.4142135623730951
[1.4142135623730951, 1.7320508075688772, 1.4142135623730951, 0.0, 2.6457513110645907, 1.7320508075688772, 1.4142135623730951, 1.4142135623730951, 2.0, 1.4142135623730951
[2.6457513110645907, 2.0, 2.6457513110645907, 2.6457513110645907, 0.0, 2.0, 2.6457513110645907, 2.23606797749979, 1.7320508075688772, 2.23606797749979]
[1.7320508075688772, 0.0, 1.7320508075688772, 1.7320508075688772, 2.0, 0.0, 1.7320508075688772, 1.7320508075688772, 1.0, 1.7320508075688772]
[1.4142135623730951, 1.7320508075688772, 1.4142135623730951, 1.4142135623730951, 2.6457513110645907, 1.7320508075688772, 0.0, 1.4142135623730951, 2.0, 1.4142135623730951
[1.4142135623730951, 1.7320508075688772, 1.4142135623730951, 1.4142135623730951, 2.23606797749979, 1.7320508075688772, 1.4142135623730951, 0.0, 2.0, 0.0]
[2.0, 1.0, 2.0, 2.0, 1.7320508075688772, 1.0, 2.0, 2.0, 0.0, 2.0]
[1.4142135623730951, 1.7320508075688772, 1.4142135623730951, 1.4142135623730951, 2.23606797749979, 1.7320508075688772, 1.4142135623730951, 0.0, 2.0, 0.0]
```

弹幕相似度矩阵（由于如果计算所有弹幕的特征向量内存会不够，故选择前 10000 行弹幕进行训练）

6.（附加）能不能对高频词（如 top 50 之类）进行可视化呈现（WordCloud 包）？

可视化函数：

```python
# 获取当前文件路径
def wordvision(data):
    #d = path.dirname(__file__) if "__file__" in locals() else os.getcwd()
    # 获取文本txt的路径（txt和代码在一个路径下面）
    #text = open(path.join(d,path),encoding='UTF-8').read()

    # 生成词云
    wc = WordCloud(
            font_path='c:\windows\Fonts\simhei.ttf',
            scale=2,
            max_font_size=100,  #最大字号
            background_color='white',  #设置背景颜色
            max_words=50
            )

    wc.generate_from_frequencies(data)  # 从文本生成wordcloud
    # wc.generate_from_text(text)  #用这种表达方式也可以

    # 显示图像
    plt.imshow(wc,interpolation='bilinear')
    plt.axis('off')
    plt.tight_layout()

    wc.to_file('C:\\Users\\dexter\\Desktop\\num_visual.jpg')  # 储存图像
    #plt.savefig('C:\\Users\\dexter\\Desktop\\num_visual.jpg',dpi=200)  #用
    plt.show()
```

对高频词的可视化呈现结果

7.（附加）能不能考虑别的特征词构建思路，如常用的 TF-IDF，即一方面词的频率要高，另一方面，词出现的文档数越少越好，观察其与仅利用词频所得的结果有何差异，哪个更好？

TF（Term Frequency，词频）是指某个词在文档中出现的次数或频率，考虑到文档有长短之分，为了便于不同文档的比较，对“词频”进行标准化。

$$\text{词频（TF）}=\text{某个词在文档中出现的次数}/\text{文档的总词数}$$

IDF（Inverse Document Frequency，逆文档频率），这是一个词语“权重”的度量，如果一个词在多篇文档中词频较低，也就表示这是一个比较少见的词，则这个词 IDF 值越大。

$$\text{逆文档频率（IDF）}=\log(\text{语料库的文档总数}/(\text{包含该词的文档数}+1))$$

将 TF 和 IDF 相乘得到 TF-IDF，基于 TF-IDF 构建如下特征词构建函数：

```python
def tfidf(content,high,one_hot):
    count1=[0 for i in range(len(high))]
    count2=[0 for i in range(len(high))]
    n=0
    for j in range(len(high)):
        for i in range(len(content)):
            if j==0:
                n+=len(content[i])
            if one_hot[i][j]!=0:
                count1[j]+=one_hot[i][j]
```

```
                count2[j]+=1

    tf_idf = [0 for i in range(len(high))]
    for i in range(len(high)):
        tf_idf[i]= count1[i]/n * math.log(len(content)/(count2[j]+1))
        if tf_idf[i]>0.01:
            print('%s,%.2f\n' % (high[i], tf_idf[i]))
```

```
小时,0.02

前,0.03

点,0.01

武汉,0.07

广告,0.01

长,0.02

哈哈哈哈,0.15
```

部分特征值权值

8.（附加）了解一下 word2vec 等深度学习中常用的词向量表征（如 gensim 和 pyltp 中均有相关的库），并思考如果用这种形式的话，那么一条弹幕会被表示成什么形式？弹幕之间计算相似性的时候，会带来哪些新的问题？

Word2Vec 是语言模型中的一种，它是从大量文本预料中以无监督方式学习语义知识的模型，被广泛地应用于自然语言处理中。

使用 genism 中的 word2vec 实现词向量的表示，保存模型后其后可以直接调用；以武汉为例查看其词向量以观察其表示形式，并输出与武汉相似度最高的 50 个词：

```python
from gensim.test.utils import common_texts, get_tmpfile
from gensim.models import word2vec
#文件位置需要改为自己的存放路径
#加载语料
sentences = word2vec.LineSentence("C:\\Users\\dexter\\Desktop\\output.txt")
#训练语料
path = get_tmpfile("word2vec.model") #创建临时文件
```

```python
model = word2vec.Word2Vec(sentences, hs=1,min_count=1,window=10)
model.save("C:\\Users\\dexter\\Desktop\\word2vec.model")
#model = word2vec.Word2Vec.load("word2vec.model") #第二次使用直接加载
#查看某个词的向量
print(model.wv['武汉'])
#输出与"武汉"相近的50个词
for key in model.wv.similar_by_word('武汉', topn =50):
    print(key)
```

运行结果如下：

```
[-0.20277311 -0.02031014  0.607445   -0.33661932 -0.42354313  0.700744
  0.6229581  -0.22375539  0.5366972  -0.69769377  0.10507049  0.43186864
 -0.94121647  0.56984484 -0.50109816  0.24032785 -0.32569575 -0.45381257
 -0.31923175 -0.6837404   0.68954045 -0.30532262  0.8006942  -0.7542101
 -0.35813284 -0.35517523 -0.45531437 -0.04194805 -0.5856913  -0.08591317
 -0.04083152 -0.09094853  0.69251996 -0.5475112   0.5071639  -0.8811151
  0.73623633  0.67668253 -0.91975856 -0.53554624  1.079305   -0.6118493
  0.644471   -0.33071834 -0.5439094  -1.2482476   0.15038426 -0.35391662
 -0.5893968  -0.09494051 -0.17471512 -0.46843922 -0.21667325 -0.6397538
  1.0041075   0.50007665 -1.1441303  -0.07099073 -1.034772    0.4063841
 -0.8451893  -0.97722036  0.12032073 -0.38704503  0.36278078  0.6300744
  0.16599815 -0.15421228  0.34063056 -0.36514118 -0.47136378 -0.7702147
 -1.1595218   0.1297642  -0.41279146  0.60749114 -0.5091267   0.07661404
  0.05273562 -0.2556926   0.42430776 -0.0628179   0.03843436 -0.92659104
 -0.05733591 -0.04864566 -0.02256352 -0.20651291 -0.5066187   0.08485667
 -0.21061197 -1.2717668  -0.13119915 -0.1870194  -0.15055981  0.32339343
 -0.1002858   0.53044254 -0.06703828  0.35266045]
```

“武汉”词向量（默认为 100 维）

```
('鸭腿', 0.39525479078292847)
('藕', 0.3854483366012573)
('碰到', 0.3742768168449402)
('想', 0.3671383857727051)
('鳥', 0.35944831371307373)
('吃', 0.35332491993904114)
('唱歌', 0.3525450825691223)
('蒜', 0.35081738233566284)
('别信', 0.3470797836780548)
('街', 0.34591513872146606)
```

输出与“武汉”相似度前 50 的词，截取前十个为例

随后考虑借助 word2vec 计算弹幕相似度，由于无法直接计算，故计算每个句子中所有单词的平均向量，然后使用向量之间的余弦相似度来计算句子相似度：

```Python
from gensim.models import word2vec
import numpy as np
from scipy import spatial

model = word2vec.Word2Vec.load("word2vec.model") #加载模型
index_to_key = set(model.wv.index_to_key)

def avg_feature_vector(sentence, model, num_features, index_to_key):
    words = sentence.split()     #将分词后的弹幕每个词分割开
    feature_vec = np.zeros((num_features, ), dtype='float32')     #初始化特征
    n_words = 0
    for word in words:
        if word in index_to_key:
            n_words += 1

            feature_vec = np.add(feature_vec, model.wv[word])     #将所有词特
    if (n_words > 0):
        feature_vec = np.divide(feature_vec, n_words)     #总和除词数得到平均值
    return feature_vec

#计算两条弹幕的平均向量
s1_afv = avg_feature_vector('进度条 警告', model=model, num_features=100, in
s2_afv = avg_feature_vector('可怕 进度条', model=model, num_features=100, in
#计算余弦相似度
```

```
sim = 1 - spatial.distance.cosine(s1_afv, s2_afv)
print(sim)
```

```
0.7707322239875793

Process finished with exit code 0
```

样例两条弹幕计算所得相似度

最后，运行 main()：

Python

```python
if __name__ == '__main__':
    main()
```