20377086 赵晟浩 第1次作业

任务：B站弹幕文本分析

任务描述：

利用弹幕文本数据（danmuku.csv，数据量约 280 万条），分词并统计词频，将每条弹幕转为向量表示，计算弹幕之间的相似度（距离），观察距离能否较好反映语义差异，能否找到最具代表性的弹幕。

附加：绘制词云图；用 TF-IDF 指标构建特征词；word2vec 的表示形式和问题

主要代码：

载入弹幕数据，用列表存储

```Python
def read_data(file_name):
    '''
    Read datafile in csv format.
    '''
    df = pd.read_csv(file_name, encoding='utf-8', usecols=[0])
    return list(df['content'])
```

载入停用词，用集合存储（主要利用到集合的 in 操作时间复杂度为 O(1)）

```Python
def make_stop_words(file_name):
    '''
    Input stop-word list into jieba module.
    '''
    stop_f = open(file_name, 'r', encoding='utf-8')
    jieba.load_userdict(file_name)
    wordset = set()
    for line in stop_f:
        line = line.rstrip('\n')
        if len(line):
            wordset.add(line)
    stop_f.close()
    return wordset
```

统计词频

counts: 词频字典

seg_list: 分词后形成的二维列表

document: 分词后用空格连接起来的原文本

if_stop_word: 该条弹幕中是否存在停用词

```Python
def word_frequency(comments, stop_words):
    '''
    Segment words with jieba and count word frequency.
    '''
    counts = {}
    seg_list = []
    documents = []
    if_stop_word = [0] * len(comments)
    i = 0
    for i in range(len(comments)):
```

```
        cur_seg_list = jieba.lcut(comments[i])
        seg_list.append(cur_seg_list)
        documents.append(' '.join(cur_seg_list))
        for seg in cur_seg_list:
            if seg not in stop_words:
                counts[seg] = counts.get(seg, 0) + 1
            else:
                if_stop_word[i] = 1
    counts = dict(sorted(counts.items(), key=lambda dc: dc[1], reverse=True))
    return seg_list, counts, documents, if_stop_word
```

提取特征词（保留出现次数大于 5 的词）

word_list: 特征集

word_id: 某词语在特征集中的序号（用于生成 one-hot 向量）

Python

```
def screen_character_word(counts):
    '''
    Extract frequently used words as characteristic words.
    '''
    word_list = [key for key, value in counts.items() if value > 5]
    word_list = list(set(word_list))
    word_id = {}
    for i in range(len(word_list)):
        word_id[word_list[i]] = i
    return set(word_list), word_id
```

随机选择不含停用词的弹幕并生成其 one-hot 向量表示

Python

```
def generate_random_comment():
    '''
    Generate a comment with at least one characteristic word

    and the corresponding 0-1 vector randomly from the list.
    '''
    id = rd.randint(0, len(seg_list) - 1)
    while if_stop_word[id]:
        id = rd.randint(0, len(seg_list) - 1)
    random_vector = [0] * (len(character_word) + 5)
    for word in seg_list[id]:
        if word in character_word:
            random_vector[word_id[word]] = 1
    return comments[id], random_vector
```

对于给定的弹幕，查找与之最接近（欧几里得距离、余弦距离）最接近的弹幕

Python

```
def search_nearest_comment(random_vector):
    '''
    Search the comment nearest to the random one either in Euclid or cosine distance.
    '''
    mind_euclid = mind_cosine = 9999.99
    comment_euclid = comment_cosine = ''
    for content in tqdm(seg_list):
        content_vector = [0] * (len(character_word) + 5)
        for word in content:
            if word in character_word:
                content_vector[word_id[word]] = 1
        if sum(content_vector) == 0 or content_vector == random_vector:
            continue
        d1 = Euclid_distance(content_vector, random_vector)
        if mind_euclid > d1:
            mind_euclid = d1
            comment_euclid = content
        d2 = cosine_distance(content_vector, random_vector)
```

```
        if mind_cosine > d2:
            mind_cosine = d2
            comment_cosine = content
    return ''.join(comment_euclid), ''.join(comment_cosine)
```

绘制 Top-n 词语的词云图

Python

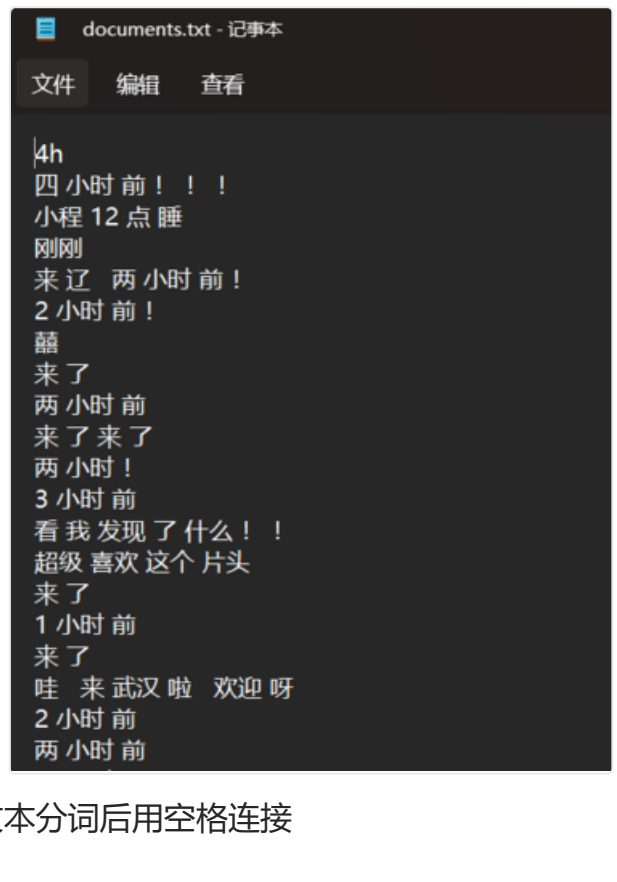
```
def word_cloud(counts, n):
    '''
    Draw wordcloud image of top-n words.
    '''
    wordpng = wordcloud.WordCloud(background_color='white', height=700, width=1000, font_path='C:\Windows\Fonts\s
    wordpng.generate(" ".join(list(counts.keys())[:n]))
    wordpng.to_file("Top_Wordcloud.png")
```

主函数：读入数据、分词并统计词频

Python

```
if __name__ == '__main__':
    comments = read_data('danmuku.csv')
    stop_words = make_stop_words('stopwords_list.txt')
    seg_list, counts, documents, if_stop_word = word_frequency(comments, stop_words)

    output_path = 'documents.txt'
    with open(output_path, 'w', encoding='utf-8') as wordfile:
        for sentence in documents:
            print(sentence, file=wordfile)
```



documents.txt 部分结果如上图所示，即原文本分词后用空格连接

输出词频统计结果，绘制词云图；

根据词频（出现次数大于 10）提取特征词；随机输出 50 个特征词

Python

```
print('TOP10词语: {}\n'.format(list(counts.keys())[:10]))
    dic_word = {'词语':counts.keys(), '次数':counts.values()}
    pd.DataFrame(dic_word).to_csv('word_counts.csv')
    word_cloud(counts, 50)

    character_word, word_id = screen_character_word(counts)
    print('词频特征词提取:', len(character_word), end='')
    for i in range(50):
        print(rd.choice(list(character_word)), end=',')
    print()
```


```
词频特征词提取：7222成本,坐等,还用,打个,赛高,江湖义气,腮帮子,家小,小粉,叫,太惨,播,诸神,这话,要活,好极了,chou,真亮,吃不来,隐藏,母上,獭蒜,全球,感受,报上,百家,多岁,红军,其他人,抚州,体会,
鹿角,内行,皿,吓死,考研,张义烈,反向,岔巴子,先别,拙劣,操,粉干,苦,怎樣,白地,嫌贵,哈哈笑,包租婆,生气,
```

随机抽取 5 条弹幕并比较距离

Python

```
random_list = []
    for i in range(5):
        content, vector = generate_random_comment()
        random_list.append(vector)
        print('随机生成弹幕', i + 1, ': ', content, sep='')
    euclid_dis = [[Euclid_distance(c1, c2) for c2 in random_list] for c1 in random_list]
    cos_dis = [[cosine_distance(c1, c2) for c2 in random_list] for c1 in random_list]
    print('随机弹幕对的欧几里得距离:\n', euclid_dis)
    print('随机弹幕对的余弦距离:\n', cos_dis)
```

```
随机生成弹幕1: 害怕
随机生成弹幕2: 武汉人会做生意
随机生成弹幕3: 错不及防
随机生成弹幕4: 天呐好听嗷
随机生成弹幕5: 卤
随机弹幕对的欧几里得距离:
[[0.0, 2.0, 2.0, 2.0, 1.4142135623730951], [2.0, 0.0, 2.449489742783178, 2.449489742783178, 2.0], [2.0, 2.449489742783178, 0.0, 2.449489742783178, 2.0], [2.0, 2.449489742783178, 2.449
489742783178, 0.0, 2.0], [1.4142135623730951, 2.0, 2.0, 2.0, 0.0]]
随机弹幕对的余弦距离:
[[0.0, 1.0, 1.0, 1.0, 1.0], [1.0, 0.0, 1.0, 1.0, 1.0], [1.0, 1.0, 0.0, 1.0, 1.0], [1.0, 1.0, 1.0, 0.0, 1.0], [1.0, 1.0, 1.0, 1.0, 0.0]]
```

从随机结果中可以看出，弹幕内容比较杂乱，基本没有语义相似，余弦距离区分度不大，而欧几里得距离差距较小，距离较小的弹幕对语义差别很大。多次随机结果没有明显改观，因此用 one-hot 向量计算弹幕相似度，效果不佳且效率较低，在特征集较大时占用很大内存。

若最有代表性的弹幕定义为句中词语在所有弹幕中出现次数多的弹幕，则可以认为是与其他弹幕平均距离最小的弹幕，理论上可以通过上述方法计算。但因这种向量表示形式只关注词的存在性而不关注词义，因此计算出的弹幕距离并不能很好反映语义差别。

TF-IDF 是常用的从文本中提取特征词的加权指标。TF 是词语在某条文档中的出现频率，IDF 是文档总数与有该词语出现的文档数量（+1 平滑处理）之比。TF-IDF 筛选出高出现频率，且低文件频率的词语，因此倾向于过滤掉常见词语，保留重要词语。采用 sklearn 库中的 TF-IDF 功能可以实现。

Python

```
def TF_IDF():
    vectorizer = TfidfVectorizer(max_df=0.5, stop_words=list(stop_words))
    X = vectorizer.fit_transform(documents)
    return vectorizer.get_feature_names_out()

names = TF_IDF()
    print('TF-IDF特征词提取:', len(names), names[-50:])
```

```
TF-IDF特征词提取: 7577 ['麻辣' '麻辣烫' '麻酱' '麻醉' '麻鸭' '黄冈' '黄字' '黄山' '黄石' '黄磊' '黄色' '黄衣' '黄衣服' '黄豆大'
 '黄酒' '黄金' '黄鳝' '黄鹤楼' '黑人' '黑店' '黑暗' '黑水' '黑点' '黑狗' '黑猫' '黑皮' '黑眼圈' '黑社会'
 '黑色' '黑药' '黑蒜' '黑说' '黑鸭' '黑龙江' '默契' '默默' '鼓掌' '鼓楼' '鼻塞' '鼻子' '鼻梁' '齐活'
 '龙之介' '龙凤胎' '龙妈' '龙子' '龙岩' '龙波' '龙虾' '龙门']
```

TF-IDF 提取的特征词略多于依据词频提取的特征词，相较而言更多的是具有代表性的名词。

word2vec 即 word to vector，将文本转化为向量表示。用向量表示单词，用矩阵表示句子。它基于 one-hot 向量的低维转化和分布式假设。word2vec 主要包括 skip-gram（由一词预测其上下文）和 CBOW（由上下文预测目标词）两种算法。

在这种形式下，单词可以被表示为较低维度的向量，而弹幕（句子）即可表示为单词对应向量组成的矩阵。

word2vec 的主要问题在于不能区分一词多义情况。

Python

```
def word_to_vec():
    sentences = word2vec.LineSentence('documents.txt')
    model = word2vec.Word2Vec(sentences, size=100, window=5, sg=0)
    return model
```