20377199 赵芮箐 第2周作业

利用python数据结构（list, dict, set等）完成简单的文本分析任务。弹幕是现下视频网站，尤其是短视频网站提供的关键功能之一。以B站为例，其有着特殊的弹幕文化，且在视频的不同部分往往会有不同话题的弹幕：比如在视频开头会出 现“来啦” “x小时前” “第一!”;在up主暗示一键三连之后常常会出现“下次一定”或者“你币有 了”;和up主建立默契之后，观众可以判断视频是否有恰饭，往往在广告之前会出现“要素察 觉” “恰饭” “快跑”等等。因此，弹幕经常被作为测度用户（viewer）与视频作者（up主）之间交互行为的关键数据。本次作业提供的数据来自B站某知名up主，已上传至课程资料的data目录下，数据格式说明如下。

a. 弹幕文件：danmuku.csv，为2799000 rows × 3 columns，本次作业仅使用第一列，即弹幕的文本内容。

b

. 停用词表示例，stopwords_list.txt

请大家尝试完成以下数据分析任务：

> 注意：不要使用jieba等库中提供的函数实现特征词抽取和文档表示，要求自己使用相关数据结构来实现；要通过函数对代码进行封装，并在main函数中调用。

任务一：处理文档

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）。

```
def read_document(filename):
    """
    读取文档函数：按行划分，转成列表返回
    """
    with open(filename, 'r', encoding='utf-8') as f:
        data = pd.read_csv(f)
        col_1 = data["content"]                          #获取第一列数据
        document_array = np.array(col_1)
        document_list = document_array.tolist()          #转成列表
    return document_list

def cut_word(lis, stopwords_filename):
    """
    分词函数：使用jieba分词，过滤停用词，返回列表
    """
    jieba.load_userdict(stopwords_filename)              #添加停用词进自定义字典

    print("----------进行分词中，请稍等----------")
    wordlist = []
    for sen in lis:
        seg_list = jieba.cut(sen, cut_all=False)         #分词
        wordlist.extend(seg_list)

    return wordlist
```

任务二：词频统计

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。

```python
def word_analyse(wordlist, stopwords_filename, n):
    """
    词频分析函数：统计wordlist中特定数目的高频词和低频词，过滤低频词，形成特征词集
    """
    with open(stopwords_filename, 'r', encoding='utf-8') as s:
        stopwords = s.read()
        stopwords_list = stopwords.split('\n')

    print("----------进行词频统计中，请稍等----------")
    word_count = {}
    for word in wordlist:                                   #过滤停用词并且统计词频
        if word not in stopwords_list:
            count = word_count.get(word, 0)
            word_count[word] = count + 1
    word_count_sorted = collections.OrderedDict(sorted(word_count.items(), key=lambda dc:dc[1],reverse=True))

    keys = list(word_count_sorted.keys())                   #观察高频词、低频词，设定词频前(后)30为高(低)频词
    print("----------高频词如下----------")
    print(keys[:30:])
    print("----------低频词如下----------")
    print(keys[-30::])
```

统计的前30个高频词和后30个低频词如下：

```
----------高频词如下----------
['哈哈哈哈', '武汉', '吃', '加油', '藕', '蒜', '好吃', '真的', '萝卜', '热情', '想', '粉', '大哥', '啊啊啊', '独头', '喜欢', '牛杂', '考古', '真', '笑', '死', '户部', '爱', '感觉', '买', '巷', '街', '树梢', '大蒜', '可爱']
----------低频词如下----------
['本本', '萎了', '电线杆', 'gao6', '盲才', '钱包', '挡住', '获取', '密码', '华贵', '指甲盖', '大小', '痛到', '双料', '康泰克', '酸爽', '加倍', '肛泰岸', '烫食', '伤食', '从浪浪', '后配', '中华', '昆明人', '要用', '表打', '可别', '写个', '电视广告', '硬推']
```

结果分析：

- 高频词

  包括B站常见弹幕，如：哈哈哈哈、啊啊啊、考古、笑死、真爱等；

  还包括和B站up相关的弹幕，如：好吃、独头蒜、树梢、大蒜等；（看出来up是**盗月社**啦）

  还包括和B站视频相关的弹幕，如：武汉、加油、藕、粉、牛杂等；（应该是去武汉的视频，还是武汉加油时期前一点的，找到了一条应该在里面的视频：【你吃过自助街吗！整条街随便点，随便坐，随便结账，就是开心！【深夜加餐饭04】】 https://www.bilibili.com/video/BV1mt411F78T?share_source=copy_web&vd_source=8674a9c5608547ccd52d1f4a57ec2e72）

  当然还是有些常见及情绪表达词汇，如：真的、热情、可爱、喜欢等。

- 低频词

  一些比较专有的、不常用、个人表达的词汇。

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。

```python
    word_feature = {}
    for key in word_count_sorted:
        if word_count_sorted[key] >= n:                     #过滤低频词
            word_feature[key] = word_count_sorted[key]
    word_pack = list(word_feature.keys())
    print("----------特征词如下----------")
    print(word_pack)

    return word_feature, word_pack
```

过滤了词频低于5000的词，获得特征词集如下：

```
----------特征词如下----------
['哈哈哈哈', '武汉', '吃', '加油', '藕', '蒜', '好吃', '真的', '萝卜', '热情', '想', '粉', '大哥', '啊啊啊', '独头', '喜欢', '牛杂', '考古', '真', '笑', '死', '户部', '爱', '感觉', '买', '巷', '街', '树梢', '大蒜', '可爱', '饿', '洛阳', '湖北', '猝不及防', '疫情', '大佬', '恰饭', '好好', '卤', '沐', '地方', '丑', '恰', '氛围', '米粉', '辣', '便宜', '希望', '长子', '四川', '牛肉', '广告', '走', '闻见', '哭', '视频', '痔疮', '江西', '邵阳', '嗦', '好像', '里', '这是', '岁', '味道', '馋', '牙膏', '超', '闻到', '恍恍惚惚', '老乡', '旁边', '弹幕', '棒', '常德', '吼吼', 'h', '闻', '香', '中国', '母上', '泪目', '气', '云南白药', '广西', '热', '去过', '红红火火', '羡慕', '跑', '太棒了', '我家', '声音', '老板', '卤菜', '手机', '确实', '肉', '烟火', '挺', '口腔溃疡', '我要', '广东', '胖', '牛', '话', '卧槽', '脆', '汤', '感动', '这条', '长', 'a', '喝', '看着', '过期', '幸福', '财大', '菜', '机会', '市场', '莲藕', '湖南', '不吃', '美食', '寂寞', '黄鹤楼', '下次', '白萝卜', '完', '俩', '宝贝', '吉庆街', '外地人', '长堤', '回来', '东西', '江湖', '玩', '真实', '锅', '北京', '吴', '长沙', '这才', '好棒', '厉害', '烟', '摄像', '掉', '肯定', '夜市', '晚上', '不行', '做', '贼', '前来', '玉子', '桂萍', '大叔', '不错', '太好了', '杨', '炒粉', '永远', '高', '吃饭', '找', '进度条', 'sao', '时间']
```

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

```
def draw_cloud(word_feature):
    """
    词云图绘制函数：用于绘制高频词的词云图
    """
    print("----------绘制词云图中，请稍等----------")
    img = Image.open("爱心.png")
    img_mask = np.array(img)
    image_colors = ImageColorGenerator(img_mask)

    wc = WordCloud(font_path="simhei.ttf", mask=img_mask, background_color="white", max_words=300, max_font_size=150)
    wc.generate_from_frequencies(word_feature)

    plt.imshow(wc.recolor(color_func=image_colors), interpolation='bilinear')
    plt.axis('off')
    plt.show()
    return
```

用特征词集做出词云图如下：

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？

```
def TF_IDF(danmulist, wordlist, stopwords_filename):
    """
    特征词获取函数：利用TF-IDF方法获取特征词
    """
    with open(stopwords_filename, 'r', encoding='utf-8') as s:
        stopwords = s.read()
        stopwords_list = stopwords.split('\n')

    print("----------进行特征词获取中，请稍等----------")
    word_count = {}
    for word in wordlist:                                #过滤停用词并且统计词频
        if word not in stopwords_list:
            count = word_count.get(word, 0)
            word_count[word] = count + 1

    word_tf = {}                                         #计算每个词的TF值
    for i in word_count:
        word_tf[i] = word_count[i]/sum(word_count.values())
```

```
    num = len(danmulist)                                #计算每个词的IDF值
    word_idf = {}
    word_danmu = collections.defaultdict(int)
    for i in word_count:
        for j in danmulist:
            if i in j:
                word_danmu[i] += 1
    for i in word_count:
        word_idf[i] = math.log(num/(word_danmu[i]+1))

    word_tf_idf = {}                                    #计算每个词的TF*IDF的值
    for i in word_count:
        word_tf_idf[i] = word_tf[i]*word_idf[i]

    # 对字典按值由大到小排序
    word_feature = collections.OrderedDict(sorted(word_tf_idf.items(), key=lambda dc:dc[1],reverse=True))

    word_pack = list(word_feature.keys())
    print(word_pack[:50:])                              #取前50个作为特征词

    return word_pack
```

得到特征词集如下：

```
['哈哈哈哈', '武汉', '加油', '吃', '藕', '好吃', '蒜', '真的', '萝卜', '热情', '啊啊啊', '想', '大哥', '考古', '独头', '喜欢', '粉', '牛杂'
, '笑', '户部', '死', '感觉', '买', '大蒜', '树梢', '巷', '可爱', '爱', '洛阳', '饿', '真', '街', '湖北', '猝不及防', '疫情', '恰饭', '大佬
', '地方', '米粉', '沐', '氛围', '丑', '好好', '便宜', '希望', '卤', '长子', '闻见', '邵阳', '牛肉']
```

结果分析：

与仅统计词频得到的特征词集进行对比分析：

```
----------特征词如下----------
['哈哈哈哈', '武汉', '吃', '加油', '藕', '蒜', '好吃', '真的', '萝卜', '热情', '想', '粉', '大哥', '啊啊啊', '独头', '喜欢', '牛杂', '考古'
, '真', '笑', '死', '户部', '爱', '感觉', '买', '巷', '街', '树梢', '大蒜', '可爱', '饿', '洛阳', '湖北', '猝不及防', '疫情', '大佬', '恰饭
', '好好', '卤', '沐', '地方', '丑', '恰', '氛围', '米粉', '辣', '便宜', '希望', '长子', '四川', '牛肉', '广告', '走', '闻见', '哭', '视频'
, '痔疮', '江西', '邵阳', '嗦', '好像', '里', '这是', '岁', '味道', '馋', '牙膏', '超', '闻到', '恍恍惚惚', '老乡', '旁边', '弹幕', '棒', '
常德', '吼吼', 'h', '闻', '香', '中国', '母上', '泪目', '气', '云南白药', '广西', '热', '去过', '红红火火', '羡慕', '跑', '太棒了', '我家',
 '声音', '老板', '卤菜', '手机', '确实', '肉', '烟火', '挺', '口腔溃疡', '我要', '广东', '胖', '牛', '话', '卧槽', '脆', '汤', '感动', '这
条', '长', 'a', '喝', '看着', '过期', '幸福', '财大', '菜', '机会', '市场', '莲藕', '湖南', '不吃', '美食', '寂寞', '黄鹤楼', '下次', '白萝
卜', '完', '俩', '宝贝', '吉庆街', '外地人', '长堤', '回来', '东西', '江湖', '玩', '真实', '锅', '北京', '吴', '长沙', '这才', '好棒', '厉
害', '烟', '摄像', '掉', '肯定', '夜市', '晚上', '不行', '做', '贼', '前来', '玉子', '桂萍', '大叔', '不错', '太好了', '杨', '炒粉', '永远
', '高', '吃饭', '找', '进度条', 'sao', '时间']
```

发现词语其实大差不差，顺序会有些区别。

像比如"真"这个词排序会下降，因为"真"这个词太普遍了，甚至可能会在一条弹幕里多次出现，因此词频很高，但其所在的文档数也会很多，故TF-IDF值会较小，顺序会下降。而像"考古"这种，会比较特别，所在的文档数不会那么多，故TF-IDF值会更大，顺序上升。

所以我觉得TF-IDF会更好一点，考虑该词出现的文档数越少越好，选出的特征词不会那么普遍，会更具有代表性。并且计算得到的TF-IDF值介于0-1之间，会更量化词的重要程度，也可以基于此来生成语料库的向量空间。

任务三：弹幕分析

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。

```python
def get_onehot_matrix(danmulist, word_pack):
    """
    得到one-hot矩阵函数：得到弹幕的onehot的矩阵（过滤掉了短的弹幕）
    """
    print("----------计算one-hot矩阵中，请稍等----------")
    #danmulist = list(set(danmulist))                    #去掉重复的弹幕
    lis = []
    danmulist_new = []
    for danmu in danmulist:                              #得到one-hot矩阵
        vector = [0] * len(word_pack)
        if len(danmu) >= 10:                             #过滤短的弹幕
            seg_list = jieba.cut(danmu, cut_all=False)
            for word in seg_list:
                if word in word_pack:
                    vector[word_pack.index(word)] = 1
            lis.append(vector)
            danmulist_new.append(danmu)
    matrix = np.array(lis)
    return danmulist_new, matrix
```

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?

- **语义相似度计算：**

```python
def danmu_compare(danmulist, matrix):
    """
    比较弹幕语义函数：随机找到一条弹幕，寻找与其语义最接近的和最远的弹幕，进行比较
    """
    index = random.randint(0,len(danmulist)-1)
    while(np.all(matrix[index]==0)):
        index = random.randint(0,len(danmulist)-1)
    print("----------随机找到的弹幕是----------")
    print("编号为:", index, "内容是:", danmulist[index])
    vector = matrix[index]

    dis_list = []
    for row in range(0, len(matrix)):                    #计算各向量与该向量的距离
        if row != index:
            dis = np.sqrt(np.sum(np.square(vector - matrix[row])))
            dis_list.append(dis)
    index_min = dis_list.index(min(dis_list))
    index_max = dis_list.index(max(dis_list))

    print("----------与其语义内容最近的弹幕内容是----------")
    if index_min < index:
        print("编号为:", index_min, "内容是:", danmulist[index_min])
    else:
        print("编号为:", index_min+1, "内容是:", danmulist[index_min+1])
    print("----------与其语义内容最远的弹幕内容是----------")
    print("编号为:", index_max, "内容是:", danmulist[index_max])
```

> PS：这里限制了随机找到的弹幕是含有特征词的，不然找到的弹幕不含特征词，即为0向量，会认为其他的不含特征词的向量与之最近，但其实语义会各不相同。这也是这个方法的一个问题。

运行结果举例：

```
----------正在比较弹幕文义----------
----------随机找到的弹幕是----------
编号为: 21701 内容是: 没加工的都不止这个价了吧
----------与其语义内容最近的弹幕内容是----------
编号为: 22543 内容是: 没加工的都不止这个价了吧
----------与其语义内容最远的弹幕内容是----------
编号为: 4711 内容是: 吉庆街旁边有生烫，有三狗，有泡蛋苕粉。都很有名。我高中三年就是这么吃胖的。
```

```
----------正在比较弹幕文义----------
----------随机找到的弹幕是----------
编号为: 2227 内容是: 户部巷，就算了。专门骗外地人的
----------与其语义内容最近的弹幕内容是----------
编号为: 619 内容是: 不去户部巷就对了，聪明
----------与其语义内容最远的弹幕内容是----------
编号为: 21 内容是: 糯藕真的一绝！！！除了清炒我爱吃脆的，别的做法都喜欢糯的
```

```
----------随机找到的弹幕是----------
编号为：  1798 内容是：  笑死我了哈哈哈哈哈
----------与其语义内容最近的弹幕内容是----------
编号为：  331 内容是：  哈哈哈哈哈哈笑死我了
----------与其语义内容最远的弹幕内容是----------
编号为：  59 内容是：  糯藕真的一绝！！！除了清炒我爱吃脆的，别的做法都喜欢糯的
```

```
----------正在比较弹幕文义----------
----------随机找到的弹幕是----------
编号为：192 内容是：卤藕真的超级奈斯
----------与其语义内容最近的弹幕内容是----------
编号为：975 内容是：卤藕世界第一，不接受反驳！！
----------与其语义内容最远的弹幕内容是----------
编号为：642 内容是：肯定是ng哥哥跟他们说的
```

结果分析：

通过计算欧几里得距离大小，找到语义最相近和相远的弹幕。

语义最近的弹幕组效果还是挺明显的，含有至少一个相同的特征词，语义也基本相似。

（发现有重复弹幕后，在获得one-hot矩阵时，用set去了个重，但仅用于分析语义相似度）

语义最远的弹幕组效果没有很明显，语义确实不相同，但这样的弹幕应该有很多，距离计算的方法会导致”最远的弹幕“一般是含有不同的特征词的弹幕，而且一般该弹幕较长。

- **找到最具代表性的弹幕：**

```python
def get_danmu_feature(danmulist, matrix):
    """
    得到最具代表性的弹幕函数：找到矩阵的重心，离重心最近的向量即最具代表性的弹幕
    """
    print("----------正在找最具有代表性的弹幕，请稍等----------")
    count = [0] * len(matrix[0])                        #计算矩阵重心
    for row in range(0, len(matrix)):
        for column in range(0, len(matrix[0])):
            count[column] += matrix[row][column]
    for column in range(0, len(count)):
        count[column] = count[column] / len(matrix)
    gravity = count

    dis_list = []
    for row in range(0, len(matrix)):                   #计算各向量与重心的距离
        dis = np.sqrt(np.sum(np.square(gravity - matrix[row])))
        dis_list.append(dis)

    dm_index_min = dis_list.index(min(dis_list))

    print("----------最具代表性的弹幕内容是----------")
    print(danmulist[dm_index_min])
```

> PS：计算所有向量重心，离重心最近的认为是最具代表性的弹幕。

运行结果举例：

- 前10000条：

```
----------最具代表性的弹幕内容是----------
来了来了！！！！最近这个速度太爽了吧
```

- 前20000条：

```
----------最具代表性的弹幕内容是----------
我黄石大冶的hhhhhhh
```

- 前50000条：

```
----------最具代表性的弹幕内容是----------
怎么到处都是这个恰饭商
```

结果分析：

前10000条最具代表性的弹幕内容是表示"来看视频了，up更新速度快"，符合B站弹幕文化。

前20000条最具代表性的弹幕内容主要是因为有个hhhhhh，B站弹幕会经常发哈哈哈哈或hhhh。

前50000条最具代表性的弹幕内容是发现该条视频是个广告视频，进行评论，符合B站发现恰饭后发弹幕的习惯。

个人觉得这个方法不是特别好，因为弹幕的话包含一整个视频时长的互动内容，内容较为分散，导致最后可能存在大量的弹幕都是0向量，所以重心会很靠近零，数据越多，内容越分散，越难得到最具代表性的弹幕。

> PS：跑这一个问如果跑全部数据的话耗时太久，故选取的是部分数据。

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？

大概的理解：

就是说我们认为，一个词的上下文的单词和它具有相似的语义，即可以理解为一个词可以被它周围的词描述。基于此，这个训练模型就是我输入一个句子的序列，然后传到隐藏层，最后再输出这个句子，不断训练这个模型。最终我们得到这个**网络的参数，就是我们语料库里所含词的向量表征**。其实目的是对输入的稀疏变量进行降维处理，与此同时相比于one-hot的表现形式，word2vec的计算给两个词语之间的相似度提供更丰富的上下文关联，因此应该可以更好地计算句子间的相似度。

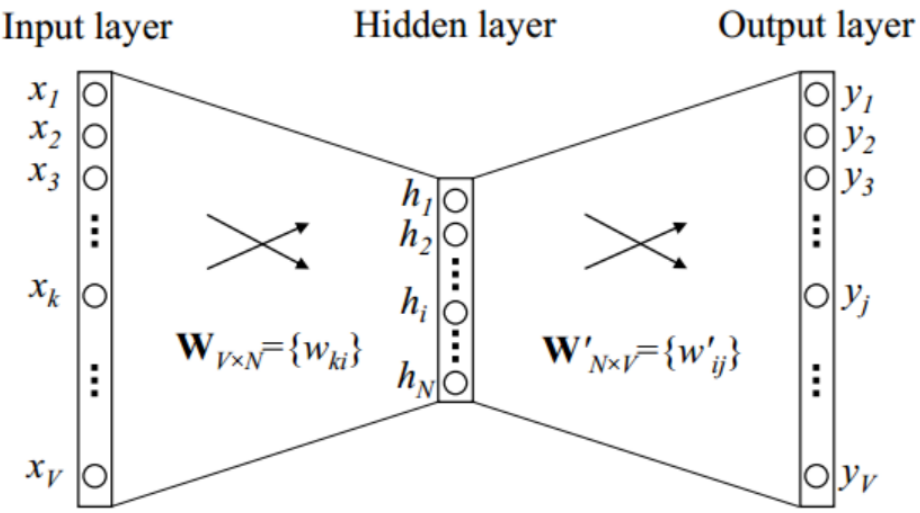

Figure 1: A simple CBOW model with only one word in the context

新的问题：

这个弹幕语料库里的词汇，都会被表示为相同维度的向量，并且这个向量间的距离表示的是**词与词之间的语义相似度**，而非句子的语义相似度。而一条弹幕会被表示为多个向量的形式（几个向量取决于含几个词），所以如果想计算句子之间的相似度，则相当于要比较的是这些不同个数的向量。

可能解决的办法：

- 每一条弹幕先提取一个自己的特征词，然后比较特征词的相似度。
- 计算一条弹幕词语的平均向量，然后再比较距离。

  或者改良一下，以每个词的TF-IDF为权重，对所有词的词向量进行加权平均，获得句子的向量表征。

  （但这个方法整体感觉很不好，损失了很多信息，可能词语完全不同，最后平均向量距离很小）

- 可以利用Word Mover算法，直接度句子之间的相似度。
- gensim库里还有doc2vec，训练结果可以得到对句子的向量表征。

代码实现：

```
def training(danmulist, stopwords):
    """
    深度学习词向量表征：用gensim中的word2vec去构建词向量
    """
    jieba.load_userdict(stopwords)                    #添加停用词进自定义字典

    with open(stopwords, 'r', encoding='utf-8') as s:
        stopwords = s.read()
        stopwords_list = stopwords.split('\n')

    print("----------进行分词中，请稍等----------")
    danmu = []
    for sen in danmulist:
        seg_list = jieba.cut(sen, cut_all=False)        #分词
        danmu.append(seg_list)

    f = open("danmulist.txt","w",encoding="utf-8")
    for line in danmu:
        for word in line:
            if word not in stopwords_list:
                f.write(word + ' ')
        f.write("\n")
    f.close()
```

```
#使用LineSentence函数处理语料，可避免前期构建语料的复杂性
model = Word2Vec(LineSentence(open('danmulist.txt', "r",encoding='UTF-8')),
                 sg=0,
                 vector_size=192,
                 window=5,
                 min_count=5,
                 workers=8
                )

#模型保存
model.save('danmu.model')

word = "粉"
print("----------输出与粉最相似的十个词----------")
if word in model.wv.index_to_key:
    print(model.wv.most_similar(word))
```

> 写这段才发现前面文档预处理函数写的不怎么好，因为前面没有返回过弹幕分词后的列表，也没存成文件。
>
> 为了这部分写的方便点，就在函数内做文件预处理了。
>
> PS：就是在我前面全部写完，看word2vec的时候看到一条说，一定养成把预处理的数据集写入文件的习惯，不然每次运行一次很耗时间，有点蠢了，下次一定。

Word2Vec参数表：

参数	作用
sg=0	使用CBOW
size	向量维度
window	windowsize
min_count=5	最小出现次数
workers	训练次数

结果展示：

```
----------输出与粉最相似的十个词----------
[('吃', 0.9962171316146851), ('藕', 0.995688259601593), ('武汉', 0.9950901865959167), ('真的', 0.9950324296951294), ('萝卜', 0.9949806332588196), ('好吃', 0.9944594502449036), ('蒜', 0.9940460920333862), ('户部', 0.993663489818573), ('嗦', 0.9930133819580078), ('牛杂', 0.9927705526351929)]
```

确实很相关！都是在“粉”前后可能会出现的词！

Ref:（打算每次把新学+参考的东西放上来）

- jieba：https://www.cnblogs.com/python-xkj/p/9247265.html
- collections：https://docs.python.org/3/library/collections.html
- 词云图：https://zhuanlan.zhihu.com/p/242740731
- TF-IDF：https://blog.csdn.net/weixin_43734080/article/details/122226507
- one-hot：https://blog.csdn.net/Robin_Pi/article/details/103732978
- word2vec：https://www.cnblogs.com/hellojamest/p/11620176.html
  https://blog.csdn.net/liaoningxinmin/article/details/122857848
- Word Mover：https://zhuanlan.zhihu.com/p/251344868