陈奕迅歌词文本分析

——大数据分析导论课程设计

任怡萌

统计学院

2016201767

在近日召开的“中国国际音乐产业大会暨 2018 第五届音乐产业高端论坛”上，《2018 中国音乐产业发展报告》发布。报告显示，2017 年中国音乐产业总规模约为 3470.94 亿元，比 2016 年增长了 6.7%，产业整体发展稳定，发展质量、增长速度好于预期[①]，音乐产业日益成为拉动中国泛娱乐消费经济的重要力量。

大陆及港澳台的音乐艺人各具才华，每一位音乐人都凭借自己独特的风格为推动音乐文化产业的发展贡献不可或缺的力量。其中，1995 年出道的陈奕迅，作为香港主流乐坛的代表性人物之一，对歌曲演绎的细腻与弹性，深厚的香港情怀，将粤语流行曲带向一个新的高度。

1 研究背景

根据报告，2017 年中国文化产业整体竞争力进一步提高，有力推动了结构优化、动力转换和质量提升，音乐产业也继续保持稳健增长，音乐原创活力进一步提升，推动产业链向多元领域延展、升级和融合。音乐的创新中，重要的一个环节是歌词的编写，挖掘优秀成名曲的歌词特征有助于新兴歌曲的创作在歌词中有卓越的成效。

本研究基于香港主流乐坛代表人物之一陈奕迅的歌词，挖掘其风格多元的歌曲特征，为创作新曲的音乐人提供宝贵建议。

2 研究问题

在文化产业发展的背景下，为了借鉴优秀的音乐创作先例，进一步促进音乐的创新发炸，本研究的主要问题主要定义为：

1. 分析陈奕迅歌词中，关注词语的相似词语是否有强烈的歌词特征；

[①] http://www.mzyfz.com/cms/benwangzhuanfang/xinwenzhongxin/zuixinbaodao/html/1040/2019-01-08/content-1379825.html

2. 对采集到的所有歌曲按照不同年份进行关键词抽取，观察歌词特征；

3. 对不同词性的词语进行分析，探究在陈奕迅歌词中，每类词性的词语有什么特征；

4. 对歌词进行分词后，在已有的文本词向量基础上训练包含歌词的新词向量，分析不同词语之间的相似度；

5. 对歌曲进行 LDA 主题建模，探究歌词的主题，观察每个主题下的关联词语都有哪些，改进仅仅使用词向量判断词语之间相关性的局限性，将陈奕迅歌词作为文档背景，挖掘更具歌手特征的文本信息；

3 数据来源及说明

本研究采集了 QQ 音乐网站中，以“陈奕迅”作为检索词搜索的结果中，193 首歌曲的所有歌词，以及每首歌曲对应的发行时间，采集程序见附件 A。

歌词数据示例如下：

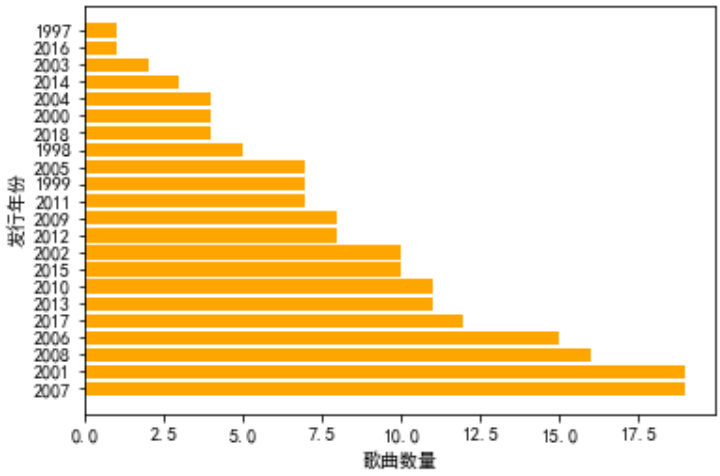
```
print(my_lyrics[0])
```

```
淘汰 (Live) - <em>陈奕迅</em> (Eason Chan)/周杰伦 (Jay Chou)\n 词：周杰伦\n 曲：周杰伦\n 编曲：C.Y.Kong\n 周杰伦：我说了 所有的谎\n 你全都相信\n 简单的 我爱你\n 你却老不信\n 你书里的剧情\n 我不想上演\n 因为我喜欢 喜剧收尾\n <em>陈奕迅</em>：我试过 完美放弃\n 的确很踏实\n 醒来了\n 梦散了\n 你我都走散了\n 情歌歌曲何必押韵\n 就算我是 K 歌之王\n 也不见得把 爱情唱得完美\n 周杰伦：只能说我输了\n 也许是你怕了\n 我们的回忆 没有皱褶\n 你却用离开烫下句点\n <em>陈奕迅</em>：只能说我认了\n 你的不安赢得你信任\n 我却得到你 安慰的淘汰\n <em>陈奕迅</em>：我试过完美放弃\n 的确很踏实\n 合：醒来了\n 梦散了\n 你我都走散了\n 情歌歌词何必押韵\n 就算我是 K 歌之王\n 也不见得把 爱情唱得完美\n 周杰伦：只能说我输了\n 也许是你怕了\n 我们的回忆 没有皱褶\n 你却用离开烫下句点\n <em>陈奕迅</em>：只能说我认了\n 你的不安赢得你信任\n 合：我却得到你 安慰的淘汰\n 周杰伦：只能说我输了\n 也许是你怕了\n 我们的回忆 没有皱褶\n 你却用离开烫下句点\n <em>陈奕迅</em>：只能说我认了\n 你的不安赢得你信任\n 合：我却得到你 安慰的淘汰\n 安慰的淘汰
```

4 描述性分析

4.1 歌曲发行时间规律

对陈奕迅所有采集歌曲的发行年份进行分析，绘制柱状图如下：



可以看出，陈奕迅发行歌曲最频繁的年份几种在 2006-2008 年，这三年也是陈奕迅推出国语、粤语专辑较多的几年，例如 2007 年粤语专辑《Listen to Eason Chan》、2008 年国语专辑《不想放手》等；图中显示，2001 年也是其创作的高峰，他发行粤语专辑《打得火热》，主打歌《K 歌之王》奠定其在歌坛的地位。

通过音乐的发行年份来看，本研究中涉及的歌词主题、关键词大多数反映了 2010 年之前陈奕迅的歌曲特征。

4.2 歌词主要内容分析

对所采集到的所有歌词进行分词，使用自定义的停用词词典，避免无意义词语的出现，并绘制词云如下（分词程序见附件 B）：

可以大致看出，陈奕迅的歌词中大多数是助词、连词、代词和动词，名词的比例偏少。其中，疑问词和否定词比较多，初步了解其歌曲的主要情感基调为比较沉郁的风格。而观察其中的名词，频率较高的出现了“怪物”、“游戏”等特殊的词语，体现陈奕迅歌词意象丰富，且同一首歌曲中重复率较高。

4.3 不同年份关键词分析

为了研究不同年份发行的歌曲中，歌词的关键词是否有差别，将所有的歌曲划分为 2009 年以前发行和 2009 年及以后发行的两部分，首先对两部分歌曲的歌词词频进行分析，绘制词云图如下（关键词分析见附件 F）：

从前后不同时期发行的歌曲歌词词云图可以看出，无论是老歌还是新歌，陈奕迅的歌词中最经常出现的总是“其实”、“没有”、“怎样”这类助词。在 2009 年之前的歌曲中，歌词是更偏具象化的，比如经常出现的名词“怪物”、“生理”、“游戏”等，而相反，在 2009 年及以后发行的新歌中，名词的词频远不及副词与助词。

对两部分歌词分别提取 TOP10 的 TF/IDF 权重最大的关键词，关键词及对应权重如下所示：

2009 年之前 TOP10 关键词（名词）

('怪物', 0.2266441652095101),
('铁箱', 0.1805643584811165),
('游戏', 0.13718564403692396),
('很漂亮', 0.13339847918765022),
('太空', 0.12640401171216178),
('寝室', 0.11616717807402448),
('土星', 0.11119661451768156),
('倒数', 0.1110775846412589),
('穿梭机', 0.11085701345941326),
('引力', 0.10903368121700369)

2009 年及之后 TOP10 关键词（名词）

('编曲', 0.11541070177934272),
('独站', 0.11225133805539905),
('树荫', 0.10391985265633802),
('坦白', 0.08914514934000001),

```
('命途', 0.06526139742723006),
('震音', 0.062007185311737094),
('掌纹', 0.06173823593046948),
('命书', 0.05770535088967136),
('感情', 0.05387811469732394),
('陷阱', 0.05385443353108921)
```

对比 2009 年前后两部分歌词中，名词词性的关键词，可以发现 2009 年以前，陈奕迅的歌词中多出现奇特、诡谲的意象，作词人喜欢将歌曲表达的情感投射于这类奇特的具象上；2009 年及以后的歌曲中，使用的关键名词不再有奇怪之感，但是整体风格渗透出孤独、伤感的感觉，这样的词风变化很可能和作词人和歌手陈奕迅的人生阅历更加丰富紧密相关。

4.4 不同词性特征分析

对名词做关键词分析后，再对所有采集歌曲中的动词和形容词做简要分析，观察其歌曲在这两类词性上的使用特点（词性分析程序见附件 F）。

在动词的使用上，陈奕迅的歌曲表现比较单一化，即经常使用若干动词如“没有”、“想到”、“发现”等，并且其动词的使用也十分通俗易懂；而形容词上，他的选择更加多样化一些，经常使用的形容词有“突然（副形词）”、“仔细”、“迟钝”、“幸福”等，陈奕迅的歌曲大多数不是直白抒发情感。二十贤惠内涵更丰富一些，从其使用的形容词就可以略知一二。

分别提取动词和形容词的 TOP10 关键词，如下所示：

动词 TOP10 关键词

('沉睡', 0.12945287774235748),
('想到', 0.08803727922500688),
('集齐', 0.08613749577802259),
('发现', 0.08550988660391076),
('自问', 0.07610453688475075),
('相仿', 0.07063180488031534),
('没有', 0.0702978607387221),
('暗示', 0.06397321893709722),
('喜欢', 0.05968007692502341),
('放出', 0.05712695486103691)

形容词 TOP10 关键词

('迟钝', 0.2370614143032702),
('灿烂', 0.17715686238941478),
('幸福', 0.1699096507263339),
('孤独', 0.16758238080433735),
('浪漫', 0.14415188834492254),
('苦困', 0.1440333434084337),
('愉快', 0.13582198649963856),
('快乐', 0.13419117352897592),
('孤单', 0.1330836164570396),
('甜蜜', 0.1267573207534854)

从两种词性的关键词可以看出，尽管陈奕迅的歌曲大多数都是情歌，但是动词的关键词并不是以情感表达的动词为主，而表达自我的一些真实举动动词更为关键；形容词上，陈奕迅的歌曲着重体现自我情感，而涉及人与人之间的情感形容词则较少。

5 文本挖掘及建模

在描述分析的基础上，为了进一步挖掘陈奕迅歌词中的文本信息，进行词向量训练与 LDA 主题建模。

5.1 词向量训练——基于已有的训练模型

由于采集到的歌词只有 193 首，为了避免数据量较小对模型训练偏差的影响，

首先使用人民日报基于其文本训练出的词向量模型，在其基础上增补陈奕迅歌词，进一步训练包含歌词的词向量（训练程序见附件 G）。

根据词向量计算两个词语之间的相似度，对比只训练歌词的模型和在人民日报词向量基础上训练歌词的综合模型，相似度比较如下：

```
print(model.similarity('幸福','孤独'))  # 综合模型
print(wv_from_text.similarity('幸福','孤独'))  # 歌词模型
0.39775902
0.3635227
print(model.similarity('幸福','浪漫'))  # 综合模型
print(wv_from_text.similarity('幸福','浪漫'))  # 歌词模型
0.27965778
0.25643036
```

对于本身词义相近的词语，只使用歌词训练的模型与综合模型的相似度差别不大；

```
print(model.similarity('十年','爱情'))  # 综合模型
print(wv_from_text.similarity('十年','爱情'))  # 歌词模型
0.25643036
0.5264515
```

对于只在歌词中有相近关系的词语（歌曲《十年》为爱情主题的歌曲），仅使用歌词训练的模型相似度计算比综合模型要更准确。

5.2 LDA 主题建模

为了进一步探究歌词的主题，观察每个主题下的关联词语都有哪些，改进仅仅使用词向量判断词语之间相关性的局限性，本研究将陈奕迅歌词作为文档背景，挖掘更具歌手特征的文本信息（建模程序见附件 H）。

对歌词分词后，建立主题模型，共建立 5 个主题，并打印每个主题对应的关键词如下所示：

```
(0,
  '0.007*"我们" + 0.007*"怎样" + 0.006*"发现" + 0.006*"没有" + 0.005*"怪物" +
0.004*"其实" + 0.004*"只知" + 0.004*"今天" + 0.003*"想到" + 0.003*"生理"'),
```

```
(1,
 '0.013*"其实" + 0.012*"怎样" + 0.011*"只知" + 0.010*"想到" + 0.009*"生理" +
0.009*"发现" + 0.009*"怪物" + 0.008*"沉睡" + 0.008*"忽然" + 0.007*"难道"'),
(2,
 '0.008*"没有" + 0.005*"其实" + 0.005*"不会" + 0.004*"什么" + 0.004*"沉睡" +
0.004*"忽然" + 0.004*"不要" + 0.004*"发现" + 0.004*"太美丽" + 0.003*"怎样"'),
(3,
 '0.010*"爱是" + 0.007*"游戏" + 0.006*"我们" + 0.006*"没有" + 0.006*"其实" +
0.005*"怀疑" + 0.004*"若能" + 0.004*"幻想" + 0.004*"不会" + 0.004*"妒忌"'),
(4,
 '0.007*"如果" + 0.005*"可以" + 0.005*"或是" + 0.005*"其实" + 0.004*"需要" +
0.004*"怎么" + 0.004*"面对" + 0.004*"难道" + 0.004*"高潮" + 0.004*"白日梦"')
```

LDA 主题模型的结果给出的 5 个主题所对应的关键词都十分相似，说明在我们采集到的陈奕迅歌曲中，通过歌词体现的主题并没有非常显著的差异，由每一个主题对应的关联词语来看，有几个非常独特的词语（如“怪物”、“生理”等），可以直接形成一个单独的主题；进一步对不同的歌曲分别找到它们对应的主题：

以歌曲《淘汰》为例：

```
0 淘汰编曲所有全都相信简单我爱你老不信书里剧情不想上演因为喜剧收尾我试完美放弃的确
踏实醒来梦散走散情歌歌曲何必押韵就算不见得爱情唱得完美只能也许我们回忆没有皱褶离开
句点只能我认不安赢得信任得到安慰淘汰我试完美放弃的确踏实醒来梦散走散情歌歌词何必押
韵就算不见得爱情唱得完美只能也许我们回忆没有皱褶离开句点只能我认不安赢得信任得到安
慰淘汰只能也许我们回忆没有皱褶离开句点只能我认不安赢得信任得到安慰淘汰安慰淘汰
        0.007*"我们" + 0.007*"怎样" + 0.006*"发现" + 0.006*"没有" + 0.005*"
怪物" + 0.004*"其实" + 0.004*"只知" + 0.004*"今天" + 0.003*"想到" + 0.003*"生
理"
```

这首歌的关键词符合主题 1（index 为 0）的主题所关联关键词的线性组合，因此《淘汰》的主题为主题 1。

6 结论

由以上的描述分析及文本挖掘与建模，可以得出结论：

1. 陈奕迅的歌曲在用词上有自己独特的风格，动词较为通俗易懂，不常用华丽的辞藻；而形容词较为含蓄委婉，并且更为多样化；整体来看，其歌曲以助

词与副词为主，名词所占比例较少，正是这样的构词方式，使得陈奕迅的歌曲更好地传达真实情感，与听众产生共鸣；

2. 从发行时间来看，由于陈奕迅是香港乐坛很早就出名的歌手，其大多数发行歌曲集中在 2006-2008 年，以 2009 年为分割点，在 2009 年之前的歌曲中，歌词是更偏具象化的，比如经常出现的名词“怪物”、“生理”、“游戏”等，而相反，在 2009 年及以后发行的新歌中，名词的词频远不及副词与助词；

3. 为了使词向量训练效果更好，本研究在已经训练好的人民日报文档词向量基础上，训练了陈奕迅歌词的词向量；与只训练歌词相比，在一般语义相近的词语中，两个模型计算相似度差异不大，但是对于只在歌词中特殊出现的词语对，歌词模型给出的相似度更为准确；

4. 陈奕迅的歌曲虽多，但是其歌词的主题并没有十分显著的差别；由于其歌曲有些歌词的意象非常独特，这些词语构成了只属于陈奕迅歌曲的主题。

7 附件

附件 A：抓取数据

```
import requests
import json
import pymongo
import time
import matplotlib.pyplot as plt
from matplotlib.pyplot import plot,savefig
import os
from os import path
import numpy as np
import jieba
import wordcloud as wc
from PIL import Image
from matplotlib import pyplot as plt
from scipy.misc import imread
import random
from collections import Counter
```

```
import string

## 抓取歌词数据
def main(page):
    print(page)
    url = 'https://c.y.qq.com/soso/fcgi-bin/client_search_cp'  # QQ 音乐搜索
链接

    data = {'qqmusic_ver': 1298,
            'remoteplace': 'txt.yqq.lyric',
            'inCharset': 'utf8',
            'sem': 1, 'ct': 24, 'catZhida': 1, 'p': page,
            'needNewCode': 0, 'platform': 'yqq',
            'lossless': 0, 'notice': 0, 'format': 'jsonp', 'outCharset':
'utf-8', 'loginUin': 0,
            'jsonpCallback': 'MusicJsonCallback4663515329750706',  # js 请求
            'searchid': '107996317060494975',
            'hostUin': 0, 'n': 10, 'g_tk': 5381, 't': 7,
            'w': '陈奕迅', 'aggr': 0
            }

    headers = {'content-type': 'application/json',
               'User-Agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
rv:22.0) Gecko/20100101 Firefox/22.0'}
    r = requests.get(url, params=data, headers=headers)
    time.sleep(3)
    # 截取出现第一个歌词的字到最后一个
    response_str = r.text
    start = response_str.find("{")
    text = response_str[start:-1]
    #     print(text)
    result = json.loads(text)
    #     print(result)
    if result['code'] == 0:
        for info in result['data']['lyric']['list']:
            item = info['content']
            song_id = info['songmid']  # 找出歌曲 id
            song_url = 'https://y.qq.com/n/yqq/song/' + song_id + '.html'  #
制作歌曲页面链接
            my_lyrics.append(item)
            song_urls.append(song_url)

my_lyrics = []
song_urls = []
```

```
for i in range(1, 21):
    main(i)

## 抓取每首歌曲的发行时间
publish_date = []
for song in song_urls:
    headers = {'content-type': 'application/json',
               'User-Agent': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
rv:22.0) Gecko/20100101 Firefox/22.0'}
    r = requests.get(song, headers = headers)
    html_text = r.text
    song_start = 'song_detail.init({\r\n\t\t\r\n\t\t\tinfo : '  # 定位歌曲信
息位置
    song_end = '\r\n\t\t\r\n\t\t});'
    index = html_text.find(song_start)
    index2 = html_text.find(song_end)
    song_info = html_text[index + len(song_start): index2]  # 提取歌曲信息
    mysong_info = json.loads(song_info)
    if 'pub_time' in mysong_info.keys():  # 判断是否有发行时间
        pub_date = mysong_info['pub_time']['content'][0]['value']
        publish_date.append(pub_date)
    else:
        publish_date.append('0000-00-00')
```

附件 B：分词 & 词频统计

```
## 分词
words = []  # 定义 list
for lyrics in my_lyrics:  # 对文本文件逐行进行分词
    words = words + jieba.lcut(lyrics)  # 分词返回 list，储存在 words 中
# 创建停用词 list，对分好的词进行筛选
text = []
stop_path = input('Your stopword path is: ')  #
E:\\renyimeng\\course\\stopwords.txt
with open(stop_path, 'rb') as f:  # 只读模式打开
    for line in f.readlines():  # 逐行读取
        text = text + line.decode().split()  # 读取后直接设置编码&按照空格分词

mywords = []
for word in words:  # 逐个筛选词语
    if word not in text:
        mywords.append(word)
```

```
## 统计词频
word_count = Counter(mywords)  # 统计词频返回 list
word_dict = dict(word_count)  # 转换为字典格式
# 搜集常见的中英文标点
zh_punctuations = '！？。“”＃＄％＆’（）＊＋，－／：；＜＝＞＠［＼］＾＿‘
｛｜｝～《》「」、、〃》「」『』【】〔〕—'
en_punctuations = '''!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'''
punctuations = zh_punctuations + en_punctuations  # 所有的中英文标点
# 去除 word_dict 里的标点符号
word_count_no_punctuations = [(word, freq) for word, freq in
word_dict.items() if word not in punctuations]  # 去除标点
word_count_long = [(word, freq) for word, freq in
word_count_no_punctuations if len(word) > 1]  # 选择长度>=2 的
word_hanzi = [(word, freq) for word, freq in word_count_long if word[0] not
in string.ascii_letters]  # 选择汉字词语
myfinal_word = [(word, freq) for word, freq in word_hanzi if word[0] not in
string.digits]  # 选择汉字词语
word_sort = sorted(myfinal_word, key=lambda x: -x[1])  # 按照词频倒序排序
# 选择词频在 top300 的词语
word_sort_200 = word_sort[0:300]
print(word_sort_200)
```

附件 C：绘制词云

```
word_dict = dict(word_sort_200)  # 转换为字典
font = 'NotoSerifCJKsc-SemiBold.otf'  # 词云的字体文件
background_Image =
np.array(Image.open("E:\\renyimeng\\course\\guitar.png"))  # 词云底图路径
img_colors = wc.ImageColorGenerator(background_Image)  # 提取底图颜色，作为词
云颜色
cloud = wc.WordCloud(
    font_path = font,
    mask = background_Image,
    scale=2,
    max_font_size = 350,  # 最大字体
    min_font_size = 30,  # 最小字体
    max_words = 300,  # 词语的数量上限
    background_color = 'white').generate_from_frequencies(word_dict)
cloud.recolor(color_func = img_colors)
plt.figure()
plt.imshow(cloud,interpolation='bilinear')
```

```
plt.axis("off")
cloud.to_file('E:\\renyimeng\\course\\word-cloud.png')  # 把词云图保存下来
```

附件 D：柱状图绘制

```
pub_year = []
for date in publish_date:
    year = date[0:4]
    pub_year.append(year)

year_count = Counter(pub_year)  # 统计词频返回 list
year_dict = dict(year_count)  # 转换为字典格式
my_year = [(word, freq) for word, freq in year_dict.items() if word not in
['0000']]  # 选择不为 0000 的发行时间
year_sort = sorted(my_year, key=lambda x: -x[1])  # 按照发行频数倒序排序
# print(year_sort)
year_freq = []
year_list = []
for year, freq in year_sort:
    year_freq.append(freq)
    year_list.append(year)

plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.barh(range(len(year_freq)), year_freq, color = 'orange', tick_label =
year_list)
plt.ylabel('发行年份')
plt.xlabel('歌曲数量')
savefig('E:\\renyimeng\\course\\pub_year.png')  # 保存绘制图片
plt.show()
```

附件 E：不同年份发行歌曲分析

```
# 每首歌分词 + 年份
lyric_word_year = list(zip(word_process, pub_year_int))
print(lyric_word_year[0])

before = [(word, year) for word, year in lyric_word_year if year < 2009]
after = [(word, year) for word, year in lyric_word_year if year >= 2009]
print(len(before))
print(len(after))
```

```
before_words = []
for (words, year) in before:
    before_words = before_words + words

after_words = []
for (words, year) in after:
    after_words = after_words + words

# 分别统计词频
before_count = Counter(before_words)  # 统计词频返回 list
before_dict = dict(before_count)
before_count_long = [(word, freq) for word, freq in before_dict.items() if
len(word) > 1]  # 选择长度>=2 的
before_sort = sorted(before_count_long, key=lambda x: -x[1])  # 按照词频倒序
排序
# 选择词频在 top300 的词语
before_sort_300 = before_sort[0:300]

after_count = Counter(after_words)  # 统计词频返回 list
after_dict = dict(after_count)
after_count_long = [(word, freq) for word, freq in after_dict.items() if
len(word) > 1]  # 选择长度>=2 的
after_sort = sorted(after_count_long, key=lambda x: -x[1])  # 按照词频倒序排
序
# 选择词频在 top300 的词语
after_sort_300 = after_sort[0:300]

# 分别绘制词云
word_dict = dict(before_sort_300)  # 转换为字典
font = 'NotoSerifCJKsc-SemiBold.otf'  # 词云的字体文件
background_Image =
np.array(Image.open("E:\\renyimeng\\course\\word_pic\\pic2.jpg"))  # 词云底
图路径
img_colors = wc.ImageColorGenerator(background_Image)  # 提取底图颜色，作为词
云颜色
cloud = wc.WordCloud(
    font_path = font,
    mask = background_Image,
    scale=2,
    max_font_size = 40,  # 最大字体
    min_font_size = 1,  # 最小字体
    max_words = 300,  # 词语的数量上限
    background_color = 'white').generate_from_frequencies(word_dict)
```

```
cloud.recolor(color_func = img_colors)
plt.figure()
plt.imshow(cloud,interpolation='bilinear')
plt.axis("off")
cloud.to_file('E:\\renyimeng\\course\\before.png')  # 把词云图保存下来

# 分别绘制词云
word_dict = dict(after_sort_300)  # 转换为字典
font = 'NotoSerifCJKsc-SemiBold.otf'  # 词云的字体文件
background_Image =
np.array(Image.open("E:\\renyimeng\\course\\word_pic\\pic5.jpg"))  # 词云底图路径
img_colors = wc.ImageColorGenerator(background_Image)  # 提取底图颜色，作为词云颜色
cloud = wc.WordCloud(
    font_path = font,
    mask = background_Image,
    scale=2,
    max_font_size = 40,  # 最大字体
    min_font_size = 1,  # 最小字体
    max_words = 300,  # 词语的数量上限
    background_color = 'white').generate_from_frequencies(word_dict)
cloud.recolor(color_func = img_colors)
plt.figure()
plt.imshow(cloud,interpolation='bilinear')
plt.axis("off")
cloud.to_file('E:\\renyimeng\\course\\after.png')  # 把词云图保存下来
```

附件 F：关键词提取

```
# 每首歌原始歌词 + 年份
lyric_year = list(zip(my_lyrics, pub_year_int))
before = [(word, year) for word, year in lyric_year if year < 2009]
after = [(word, year) for word, year in lyric_year if year >= 2009]

before_lyrics = ''
for lyric, year in before:
    before_lyrics = before_lyrics + lyric
after_lyrics = ''
for lyric, year in after:
    after_lyrics = after_lyrics + lyric
jieba.analyse.extract_tags(before_lyrics, topK = 5, withWeight = True,
allowPOS = ('n'))
```

```
# 不同词性关键词
all_lyrics = ''
for lyric, year in lyric_year:
    all_lyrics = all_lyrics + lyric
print(jieba.analyse.extract_tags(before_lyrics, topK = 10, withWeight =
True, allowPOS = ('n')))
print(jieba.analyse.extract_tags(after_lyrics, topK = 10, withWeight =
True, allowPOS = ('n')))
print(jieba.analyse.extract_tags(all_lyrics, topK = 10, withWeight = True,
allowPOS = ('v')))
print(jieba.analyse.extract_tags(all_lyrics, topK = 10, withWeight = True,
allowPOS = ('a')))
```

附件 G：词向量训练

```
mywords = []
for sentence in my_lyrics:
    temp = jieba.lcut(sentence)
    mywords.append(temp)
word_process = []
for words in mywords:
    temp = []
    for word in words:
        if word not in punctuations:
            if len(word) > 1:
                if word[0] not in string.ascii_letters:
                    if word[0] not in string.digits:
                        if len(word) > 0:
                            if word not in text:
                                temp.append(word)
    if len(temp) > 0:
        word_process.append(temp)

model = gensim.models.Word2Vec(size=300, min_count=1)
model.build_vocab(word_process)
# 加载已有的词向量
PRE_TRAINED_WORD2VEC = 'E:\\renyimeng\\course\\data\\sgns.wiki.word.bz2'
wv_from_text = KeyedVectors.load_word2vec_format(PRE_TRAINED_WORD2VEC,
binary=False)  # 仅使用歌词训练
# 重新训练综合模型
model.build_vocab([list(wv_from_text.vocab.keys())], update = True)
model.intersect_word2vec_format(PRE_TRAINED_WORD2VEC, binary = False, lockf
```

```
= 1.0)
# 再训练
model.train(word_process, total_examples = model.corpus_count, epochs =
model.epochs)
# 进行相关性比较
print(model.similarity('幸福','孤独'))  # 综合模型
print(wv_from_text.similarity('幸福','孤独'))  # 歌词模型
print(model.similarity('幸福','浪漫'))  # 综合模型
print(wv_from_text.similarity('幸福','浪漫'))  # 歌词模型
print(model.similarity('十年','爱情'))  # 综合模型
print(wv_from_text.similarity('十年','爱情'))  # 歌词模型
```

附件 H：LDA 主题建模

```
train = []
stopwords =
open('E:\\renyimeng\\course\\stopwords.txt','r',encoding='utf8').readlines(
)
stopwords = [ w.strip() for w in stopwords ]
for line in my_lyrics:
    line = jieba.lcut(line.rstrip())
    train.append([ w for w in line if w not in stopwords and w not in
punctuations and len(w) > 1 and w[0] not in string.ascii_letters and w[0]
not in string.digits and w not in stopwords])
dictionary = corpora.Dictionary(train)
print(dictionary)

corpus = [ dictionary.doc2bow(text) for text in train ]
corpus_tfidf = models.TfidfModel(corpus)[corpus]
lda = models.LdaModel(corpus, num_topics = 5, id2word = dictionary)  # 设定
主题数为5
lda.print_topics(5)

# 查看每首歌曲所属的主题
topicList = lda.print_topics(5)
i=0
for doc in corpus_tfidf:
    topics = lda.get_document_topics(doc)
    topic = 0
    max_weight=0
    for t in topics:
        # topics 是这篇文章的所有 topic
        if t[1]>max_weight:
```

```
            topic = t[0]  # 保存权重最大的话题id
            max_weight = t[1]
    print ( str(i) +"\t" +str(topic) + "".join(train[i]) +"\t" +
topicList[topic][1]  +"\t"  +"\n")  # 输出为【粘连的句子】+这个句子相关的话题，
粘连句子之前的index为它在训练集的位置
    i= i + 1
```