深圳大学考试答题纸

(以论文、报告等形式考核专用)

二〇一九～二〇二〇 学年度第二学期

课程编号 课程名称 文本大数据分析与挖掘 主讲教师 卢亚辉 评分

学 号 2018191118 姓名 潘海飞 专业年级 2018 级数学与应用数学

教师评语：

题目： 《我和我的祖国》电影评论分析

算法演示+PPT 讲解成绩	大作业报告成绩

一：网络爬虫

本报告爬取 2019 年国庆期间的热播电影《我和我的祖国》的用户评论，爬取的内容包括评论人姓名、评论时间以及评论内容。爬取过程中采用的 python 第三方库是 requests、BeautifulSoup。通过观察发现，《我和我的祖国》的每一页电影评论都有规律，不同的页面仅仅在某个数字上有差异，所以我们设置 while 循环爬取多页评论数据，每次循环 k 都增加 20，代表访问下一页的评论。在爬取数据的过程中，设置 name_list, time_list, comment_list 三个最关键的列表，采用 while 循环，利用 append 方法向列表增加数据。爬取所有页面数据后，采用 dataframe 把之前的列表综合存储为 comment_df，并保存成 csv 文件。以下为部分 csv 文件截图

	A	B	C	
1		Name	Time	Comment
8	6	_	2019/9/30 1:31	几个人一起看，前面排名无所谓，最差大家都很一致——白昼流星 因此扣了一星
9	7	西楼尘	2019/9/28 21:20	旗杆上升起的不是一块红布，报纸上登载的不是一个名字，电视外错过了情窦，鸟巢里交换了运气。秒针精确了百年的等待，流星划破了回乡的急
10	8	胤祥	2019/9/29 7:06	7场"最后一分钟营救"，比的就是怎么去做这个最后一分钟营救。7部评分及个人喜好排序：管虎(4)>徐峥(4)>宁浩(3)>张一白(3)>薛晓璐(2)>文牧野(2
11	9	米米路亚0815	2019/9/30 9:29	最喜欢女排那个故事，小演员太可爱了。 优叔一出现感觉过年了。
12	10	悠三岁	2019/9/28 22:46	宁浩是最好的，葛大爷是完美的。
13	11	陈一	2019/9/28 18:37	陈凯歌凭一己之力把宁浩拉上去的一星又还回去了。
14	12	影志	2019/9/28 22:30	谁能想到呢？这可能是国庆档最好看的一部。从片头王菲主题曲开始就被渲染，看之前有主旋律的刻板印象，但当故事从一个个小人物的切口开始
15	13	Ston3s	2019/9/28 22:54	1，剧本稀烂导演喝醉了，但演员都特别在线，故事原型太棒，以致看预告我就哭傻了。2，任素汐的感觉一直狠好，张嘉译真帅啊啊啊啊。切尔诺
16	14	掉线	2019/9/30 0:09	这才是主旋律电影正确的打开方式啊，没有一味强调家国情怀，几乎都是以普通人，小人物视角去切入，共鸣感和好看程度一下就上来了。戏剧化
17	15	柯里昂	2019/9/29 22:19	功绩和成就需要被歌颂，主旋律电影存在的合理性也从未被质疑过，与此同时如果能一并回头看看那些被遗忘的和没被正视过的苦痛，或许才是真
18	16	糖小包	2019/9/30 0:00	我以前是不怎么感冒献礼片的，这次因为朱一龙的缘故关注了我和我的祖国，才发现我比自己想象中更爱这个国家。这部献礼片的意义，是串联，
19	17	黄青蕉	2019/9/29 17:02	任务型电影拍成这样可以了，难得的大时代下看到一点小人性，徐峥（居然）最佳，宁浩第二，其他正常发挥，凯歌老师…emmmmm儿子特写拍大
20	18	小雏菊的刺	2019/9/30 0:02	《我和我的祖国》回归篇真实再现了香港回归的场景，青年演员朱一龙在其中饰演了一名护旗手，当他迈着坚定的步伐走向升旗台时，我好像也到
21	19	山鬼	2019/9/28 18:11	《我和我的祖国》由七位导演各执一环、接年代事件串接，虽风格各异但共成一色，十分特别。与前几个故事或昂扬或细腻或幽默不同，由陈凯歌

二、分词并去除停用词

本报告采用 jieba 进行分词操作，jieba 中文表述为“结巴”，是用 Python 编写的最好用的中文分词组件之一，jieba.cut 是 jieba 模块下进行中文语句分词的主要函数，调用方式为：jieba.cut(sentence,cut_all=False)，sentence 表示需要分词处理的字符串，cut_all:分词模式，True 代表全模式，False 代表精确模式。

之前我们设置的列表 comment_list 存储了所有用户的评论，所以我们可以设置 for 循环，对 comment_list 里面的每条数据，采用 jieba.cut(comment_list[i],cut_all=False)进行分词，对于已经分割好的句子，我们导入 stopwords，查看分好的词是否在 stopwords，若出现在 stopwords 上，则去处本词。同样的，我们设置 comment_part_list 列表采用 append 方法存储每条评论分词之后的词语，最后把 comment_part_list 加入 comment_df，从而完成实验数据准备阶段的所有操作，即爬取评论、对评论进行分词并去除停用词，最后所有数据存储于 comment_df。

In [5]:
```
comment_part_list =[]
stopwords = [line.strip() for line in open('stopwords.txt','r',encoding='utf-8').readlines()]
for i in range(len(comment_list)):
    sentence_seged = jieba.cut(comment_list[i],cut_all=False)
    final = ''
    for word in sentence_seged:
        if word not in stopwords:
            if word != '\t':
                final +=word
                final +=" "
    comment_part_list.append(final)
```

三、文本特征工程

3.1 制作词云图

词云图将关键词按照一定的顺序和规则排列，以文字的大小代表词语的重要性，直观、快速地展示重要文本信息。wordcloud 是较为常用的 Python 词云绘制包，利用其定义的 WordCloud 类即可实现词云图的绘制。

我们将之前存储的 comment_part_list 导入函数，即可获得如下词云图。

3.2 文档-词项矩阵

词袋模型（BOW），是构建文档-词项矩阵的基本思想。对于给定的文本，可以是一个段落，也可以是一个文档，该模型都忽略文本的词汇顺序和语法、句法，假设文本是由无序、独立的词汇构成的集合，这个集合可以被直观的想象成一个词袋，袋子里面就是构成文本的各种词汇。

本报告采用 sklearn.feature_extraction.text 中的 CountVectorizer 制作文档-词频矩阵：

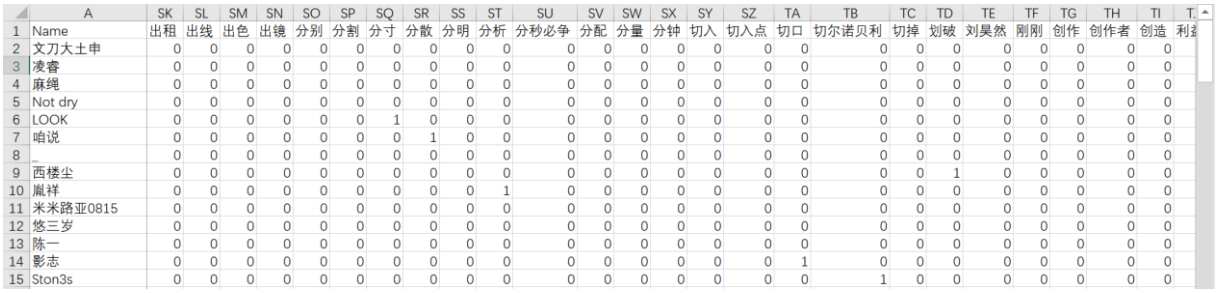
sklearn.feature_extraction.text 是 sklearn.feature_extraction 包中进行文本数据结构化处理的模块，其中定义的 CountVectorizer 类可以同时实现分词处理和词频统计，并得到文档-词频矩阵，把之前得到的 comment_part_list 转化为 ndarray 之后，我们可以直接输入参数得到文档-词项矩阵。直观来看，矩阵的行代表文档，列代表词汇，矩阵元素即为文档中某一词汇出现的次数。

In [7]:
```
from sklearn.feature_extraction.text import CountVectorizer
comment_part_ndarray = np.array(comment_part_list)
statistics_bwm = CountVectorizer(min_df=0., max_df=1.)
comment_vectorization_bwm_ndarray = statistics_bwm.fit_transform(comment_part_ndarray).toarray()
```

In [9]:
```
comment_vectorization_bwm_ndarray
```

Out[9]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 1, 0]], dtype=int64)
```

得到文档-词项矩阵后，我们可以利用 statistics_bwm.get_feature_names() 所有的得到列名，构建 DataFrame：comment_vectorization_bwm_df，其中每一行代表一个用户的评论，每个格里面的数字代表这一列名的词语在本评论中出现的次数，最后保存成“评论-词项矩阵.csv”，



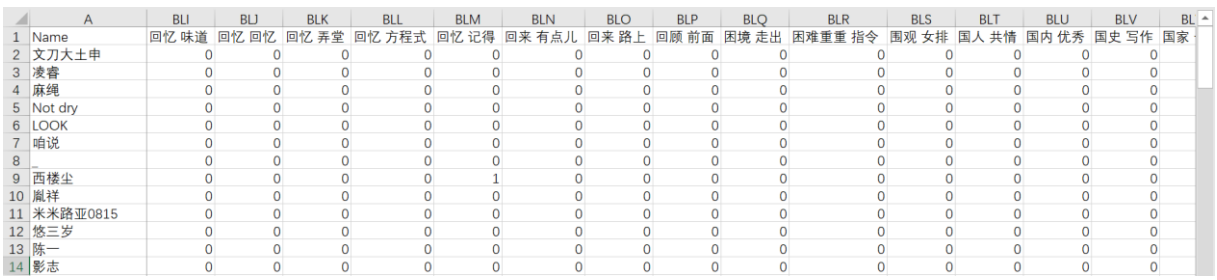
	A	SK	SL	SM	SN	SO	SP	SQ	SR	SS	ST	SU	SV	SW	SX	SY	SZ	TA	TB	TC	TD	TE	TF	TG	TH	TI	T
1	Name	出租	出线	出色	出镜	分别	分割	分寸	分散	分明	分析	分秒必争	分配	分量	分钟	切入	切入点	切口	切尔诺贝利	切掉	划破	刘昊然	刚刚	创作	创作者	创造	利
2	文刀大土申	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	凌睿	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	麻绳	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	Not dry	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	LOOK	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
7	咱说	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
9	西楼尘	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
10	胤祥	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
11	米米路亚0815	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
12	悠三岁	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
13	陈一	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
14	影志	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
15	Ston3s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	

虽然文档-词项矩阵没有考虑到词汇之间的依存关系，但是这一简单假设也大大简化了后续文本挖掘的计算过程，利用结构化处理的文档-词项矩阵已经可以实现很多有意义的分析过程，如计算文档之间的相关性、文本分类、文本聚类等等。

然而，词袋模型的一个缺点就是其忽略了语句中词汇的顺序及语法关系，如以下两句话“北京中国首都”、“中国首都北京”两句话由相同的词汇组成，文档-词频矩阵中对应的向量表示也应该是一样的，但是表达的语意不同。

```
In [40]: corpus = ['北京 中国 首都',
         '中国 首都 北京']
         vectorizer = CountVectorizer()
         X = vectorizer.fit_transform(corpus)
         X.toarray()

Out[40]: array([[1, 1, 1],
                [1, 1, 1]], dtype=int64)

In [41]: bigram_vectorizer = CountVectorizer(ngram_range=(1, 2))
         X_2 = bigram_vectorizer.fit_transform(corpus).toarray()
         X_2

Out[41]: array([[1, 1, 1, 1, 1, 0],
                [1, 1, 1, 0, 1, 1]], dtype=int64)

In [42]: bigram_vectorizer.get_feature_names()

Out[42]: ['中国', '中国 首都', '北京', '北京 中国', '首都', '首都 北京']
```

为了避免丢失这部分信息，我们可以用 1-grams（单个词语）和 2-grams（两个词语组成的词组）的分词方法来构建文档-词频矩阵，我们需要更改的输入参数只是 CountVectorizer(ngram_range=(1, 2)) 其余与之前的操作类似，构建 DataFrame：comment_vectorization_b2gm_df，其中每一行代表一个用户的评论，每个格里面的数字代表这一列名的词语在本评论中出现的次数，最后保存成“评论-词项矩阵（双词）.csv”。



	A	BLI	BLJ	BLK	BLL	BLM	BLN	BLO	BLP	BLQ	BLR	BLS	BLT	BLU	BLV	BL
1	Name	回忆 味道	回忆 回忆	回忆 弄堂	回忆 方程式	回忆 记得	回来 有点儿	回来 路上	回顾 前面	困境 走出	困难重重 指令	围观 女排	国人 共情	国内 优秀	国史 写作	国家
2	文刀大土申	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	凌睿	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	麻绳	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	Not dry	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	LOOK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
7	咱说	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
9	西楼尘	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
10	胤祥	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
11	米米路亚0815	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
12	悠三岁	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
13	陈一	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
14	影志	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

3.3 词频-逆向文档频率（TF-IDF）

在大量的文本数据中，通常会存在一些出现频率极高但是并无实际意义的词汇，如部分停用词，如果直接用这些所谓的高频词对文档进行进一步的分析处理很可能会忽略某些出现频率没有那么高的重要词汇，所以需要在词频的基础上对各个词汇的频数进一步调整，“词频-逆向文档频率”就是一种常用的调整方式。

该算法的原理如下：$\text{tf-idf(w)} = \text{tf(d,w)} \times \text{idf(w)}$，其中 tf(d,w) 代表词频，$\text{idf(w)} = \log(\frac{N}{N(w)})$

变量说明：
N 为语料库中文档的总数，在本报告中，即为评论的数量。
N(w)是词语 w 出现在多少个文档中

我们可以从该算法的公式得知并不是出现的越多就越重要，同时并不是出现的越少就越不重要！打个比方，加入某一词语在每条评论中都出现，那么按照 TF-IDF 的算法原理，该单词的重要程度大大降低，而假如某一词语仅仅在一条评论中出现，那么该单词有着独特性，idf(w)会非常大，从而导致 tf-idf(w)变大，提高了该单词的重要程度。

本报告采用 sklearn.feature_extraction.text 中的 TfidfVectorizer ，TfidfVectorizer 类可将原始文本数据直接转换为 TF-IDF 矩阵

```
from sklearn.feature_extraction.text import TfidfVectorizer

statistics_tv = TfidfVectorizer(min_df=0., max_df=1., norm='l2',use_idf=True, smooth_idf=True)
comment_vectorization_tv_ndarray = statistics_tv.fit_transform(comment_part_ndarray).toarray()
comment_vectorization_tv_ndarray
```

```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.20659331,
        0.        ]])
```

得到文档-词项矩阵后，我们可以利用 statistics_tv.get_feature_names() 所有的得到列名，构建 DataFrame：comment_vectorization_tv_df，其中每一行代表一个用户的评论，每个格里面的数字代表这一列名的词语在本评论中的 tf-idf 系数，最后保存成“词频-逆向文档频率(TF-IDF Model）.csv”。

	KS	KT	KU	KV	KW	KX	KY	KZ	LA	LB	LC	LD	LE	LF	LG	LH	LI	LJ	LK	LL	LM	LN	LO	LP	LQ
1	产生共鸣	京片子	亮点	亮相	亮眼	亲历	亲历者	亲身经历	人世间	人去	人和事	人唱	人多	人心	人性	人文	人民	人民共和国	人潮	人物	人理	人类	仁慈	仅供	今天
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.11	0.11	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0.13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

四、文本相似度分析

4.1 提取新文本的特征向量

很多时候，我们需要对新的评论进行特征分析，如果每次都要对评论进行 jieba 分词、去除停用词再来构建文本向量，其实这是很耗时间的。那么，有没有一种方法可以自动帮我们提取一条新评论的特征，直接返回新评论的特征向量呢？其实，我们之前已经用爬虫得到的评论作为训练样本提取出每条评论的特征向量，而 Scikit-Learn API 提供了 transform 函数，该函数可以根据我们之前提供的训练样本，当我们输入一条新评论时，该函数可以直接返回该新评论的特征向量。

```
In [80]: new_doc = '香港 祖国 红色 五星红旗 团结 感人 苦了 中华儿女 2008 陈凯歌 徐峥 黄渤 男孩 奥运 女排'

In [81]: new_comment_vectorization_tv_df = pd.DataFrame(np.round(statistics_tv.transform([new_doc]).toarray(), 2),
                                                         columns=statistics_tv.get_feature_names())
         new_comment_vectorization_tv_df.to_csv('新评论特征向量提取.csv',encoding="utf_8_sig")
         new_comment_vectorization_tv_df
```

Out[81]:

	00	08	10	15	155	16	20	2008	2009	21	...	黄渤	黄渤宋佳	黄色	黑色幽默	默默无闻	黯淡无光	鼓掌	鼻子	齐聚	齐飞
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.37	0.0	0.0	...	0.32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

1 rows × 2689 columns

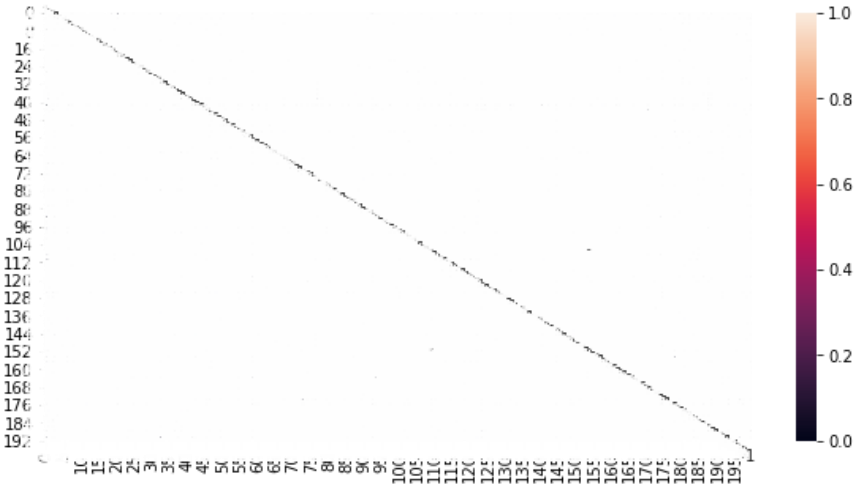
4.2 计算相似度矩阵并可视化

我们之前已经得到了所有评论的词频-逆向文档频率矩阵（TF-IDF），现在，我们想分析这些评论的相关度，其实之前我们已经做了很多工作了，我们已经把每条评论转化成一个空间一个 n 维向量，一个很简单的方法是计算这些点之间的空间欧式距离，当这个距离越小，可以认为这两个点之间越靠近，从而得出这两条评论的相关系数。在 sklearn.metrics.pairwise 中，提供了 cosine_similarity 函数，当我们把之前的词频-逆向文档频率矩阵，即 comment_vectorization_tv_ndarray，作为输入参数给 consine_similarity 函数，返回的则是一个相似度矩阵，把这个矩阵保存成“评论相似度矩阵.csv”，如下图所示：

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		0	1	2	3	4	5	6	7	8	9	10	11
2	0	1	0.07457103	0.034785818	0	0.012608503	0.008748275	0.104577958	0	0.008246337	0	0	0.033349192
3	1	0.07457103	1	0	0	0.038410059	0.020008527	0.041424487	0.056017761	0.016271145	0	0	0.009853313
4	2	0.034785818	0	1	0	0.030460715	0.02113484	0	0	0.008816774	0	0	0.035656107
5	3	0	0	0	1	0	0	0	0	0	0	0	0
6	4	0.012608503	0.038410059	0.030460715	0	1	0.063299388	0	0.03491778	0.065813635	0	0	0.027485603
7	5	0.008748275	0.020008527	0.02113484	0	0.063299388	1	0.015586321	0.008841939	0.05625536	0.020894281	0.023754664	0.019070591
8	6	0.104577958	0.041424487	0	0	0	0.015586321	1	0.011335206	0	0	0	0.127327871
9	7	0	0.056017761	0	0	0.03491778	0.008841939	0.011335206	1	0	0.024324697	0	0
10	8	0.008246337	0.016271145	0.008816774	0	0.065813635	0.05625536	0	0	1	0.013130306	0	0.02750016
11	9	0	0	0	0	0	0.020894281	0	0.024324697	0.013130306	1	0	0
12	10	0	0	0	0	0	0.023754664	0	0	0	0	1	0
13	11	0.033349192	0.009853313	0.035656107	0	0.027485603	0.019070591	0.127327871	0	0.02750016	0	0	1
14	12	0.010857785	0.07179567	0.026231177	0	0.047600652	0.072899946	0.021199879	0.039279339	0.100133025	0.077426477	0	0.023669167
15	13	0	0.011026695	0	0	0.035857023	0.054661678	0	0	0.023129187	0.07626464	0.05225349	0

通过简单地观察，对角线之间的相关系数都为 1，不难理解，对角线代表的是本条评论自身的相关系数，而其余评论之间的相关度都非常小，多数都小于 0.1，我们可以简单地进行推断，该实验爬取的评论样本具有较高的原创性，并没有大面积地出现复制他人评论的现象。

为了更好地对相似度矩阵进行分析，可以利用 seaborn 库，利用 heatmap 函数，画出相似度矩阵热力图。



五、评论聚类（K-means 实现）

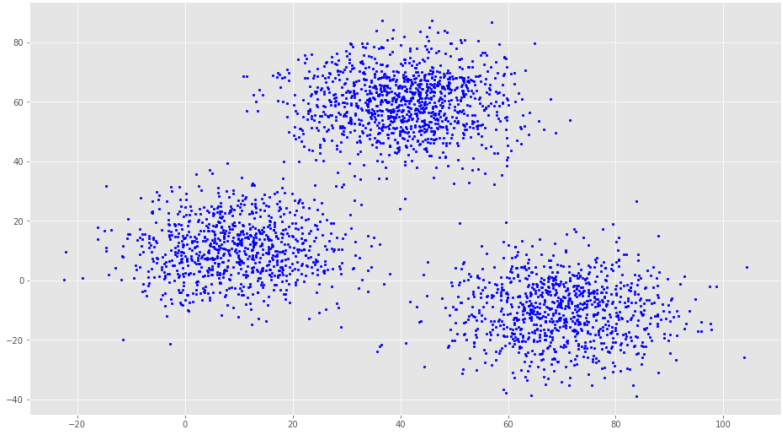
4.1 K-mean 算法的原理：

1. 设置初始类别中心和类别数;
2. 根据类别中心对全部数据进行类别划分：每个点分到离自己距离最小的那个类;
3. 重新计算当前类别划分下每个类的中心;
4. 在得到类别中心下继续进行类别划分;
5. 如果连续两次的类别划分结果不变则停止算法；否则循环 2～5 ;

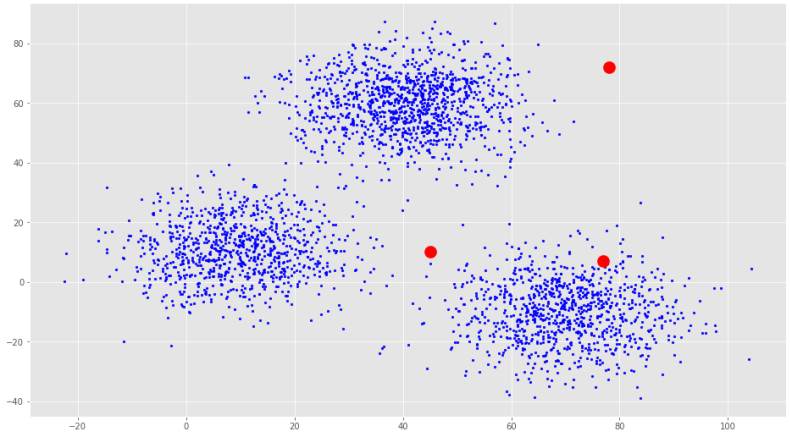
4.2 K-mean 算法代码实现（详细代码请看 jupyter notebook 文件）：

为了更好地理解 K-mean 聚类算法的实现过程，本报告以 3000 个二维样本坐标（具体见文件“k-mean.csv”）为示例，讲解 k-mean 聚类算法的运行过程。

首先，读取 k-mean.csv 文件，利用 plt.scatter 对 3000 个二维坐标进行散点图可视化



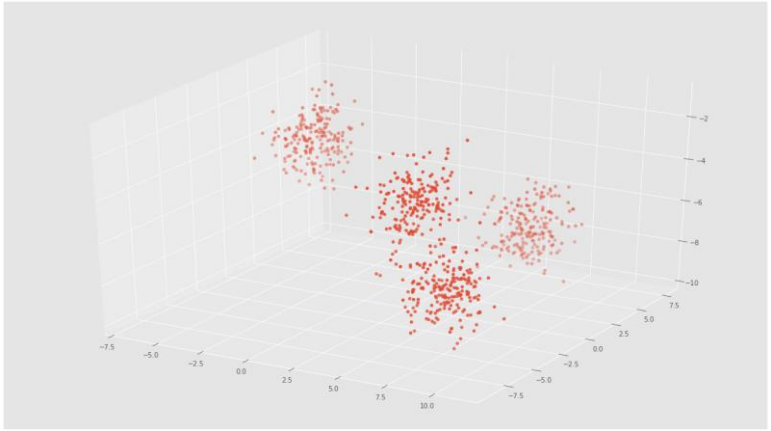
步骤一：设置类别中心和类别数，我们认为设置类别数为 3 类，numpy 里面的随机函数 np.random.randint 完成 3 个中心点的随机选择，并用红色标记初次随机选择的中心点。

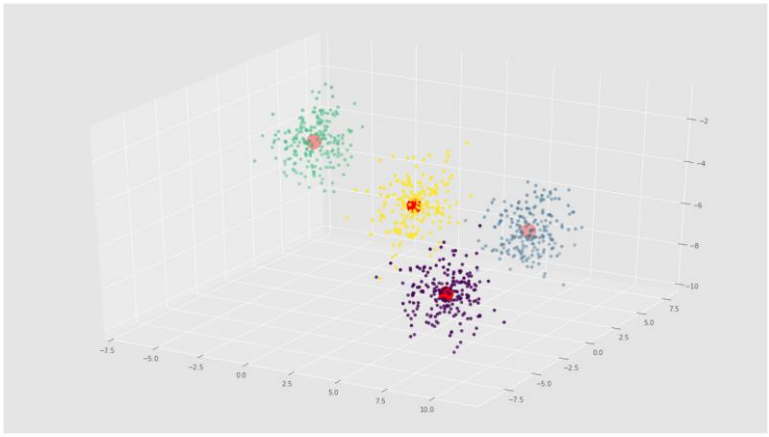


步骤二：1：根据类别中心对全部数据进行类别划分：每个点分到离自己距离最小的那个类；2：重新计算当前类别划分下每个类的中心; 3：在得到类别中心下继续进行类别划分;

如果连续两次的类别划分结果则停止算法，否则循环步骤 2。

类似的，我们同样可以对 n 维空间的点运用 k-mean 算法进行聚类，下面给出 3 维空间的示例图。





有前面 K-mean 算法的理论基础，把前面的 2 维空间、3 维空间拓展到之前 TF-IDF 矩阵的列数，把每条评论看作是空间上的一个点，同样也对评论进行聚类操作，本报告把之前的的评论分为 2 类，可以近似认为一类是正评价，另一类是负评价，sklearn 库 cluster 包中的 kmeans 模块下定义了实现 k-means 聚类的类 KMeans 及相关方法，对评论进行聚类，使用 kmeans.cluster_centers_可以得出两个中心点的坐标。

```
In [219]: from sklearn.cluster import KMeans
          import numpy as np
          kmeans = KMeans(n_clusters=2)

In [220]: kmeans.fit(comment_vectorization_tv_ndarray)

Out[220]: KMeans(n_clusters=2)

In [222]: kmeans.cluster_centers_

Out[222]: array([[ 1.23210351e-03,  4.67226238e-03,  2.48980262e-03, ...,
                   2.07170967e-03,  1.33286003e-03,  1.01427558e-03],
                 [ 8.67361738e-19,  3.90312782e-18,  1.95486390e-03, ...,
                  -6.50521303e-19,  6.50521303e-19, -3.25260652e-19]])
```

有了前面的 kmeans 类，可以使用 kmeans.inertia 查看 inertias，表示样本距离最近的聚类中心的总和，它是作为在没有真实分类标签下的非监督式评估指标，该值越小越好，值越小证明样本在类间的分布越集中，即类内的距离越小。还可以使用 kmeans.fit_predict 对评论的属性进行预测。

```
In [223]: kmeans.inertia_

Out[223]: 190.96202301406782

In [224]: kmeans.fit_predict(comment_vectorization_tv_ndarray)

Out[224]: array([1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0,
                 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
                 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0,
                 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
                 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
                 1, 1])
```

4.3 聚类效果评价

sklearn 库 metrics 模块下的 cluster 包给出了多种聚类效果评价指标，如果已知聚类对象的实际类别标签（可以用人工分别标签），结合预测的分类结果，就可以计算调整兰德指数对比二者的相似度，对于一个样本量为 n 的样本空间 S ，我们将空间中原本的样本标签划分与我们要评价聚类结果划分分别用 X={X1,X2,...,Xn} 与 Y={Y1,Y2,...Yn} 表示，此时调整兰得指数 R 可以由如下公式表示：

$$R=\frac{a+b}{a+b+c+d}=\frac{a+b}{\binom{n}{2}}$$

其中，上式中的指标 a,b,c,d 含义如下：

a:在样本空间 S 中，在 X 划分中属于同一类且在 Y 划分中属于同一类的样本对数
b:在样本空间 S 中，在 X 划分中属于不同类且在 Y 划分中属于不同类的样本对数
c:在样本空间 S 中，在 X 划分中属于同一类且在 Y 划分中属于不同类的样本对数
d:在样本空间 S 中，在 X 划分中属于不同类且在 Y 划分中属于同一类的样本对数

In [242]:
```
from sklearn import metrics
labels_true = [0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       1, 1]
labels_pred = list(kmeans.fit_predict(comment_vectorization_tv_ndarray))
metrics.adjusted_rand_score(labels_true, labels_pred)
```

Out[242]: 0.569212225639797

可以看到，上式中的指标 a+b 的值越大，反映了划分 X 与 Y 之间的一致程度越强；而指标 c+d 的值越大，则反映了划分 X 与 Y 之间的差异程度越高。调整兰德指数正是以这样一种直观的比值形式来反映聚类结果与样本标签的相似程度。该函数直接返回取值范围在-1 到 1 之间的聚类结果相似度得分，如果结果位于 0-1 之间表明聚类效果较好。

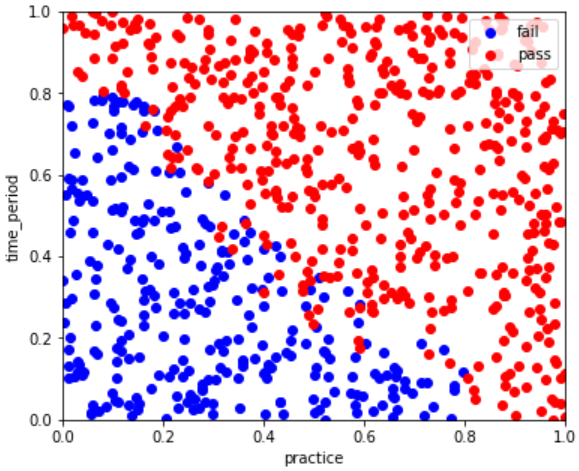
五：评论分类（SVM 算法实现）

5.1 SVM 算法原理：

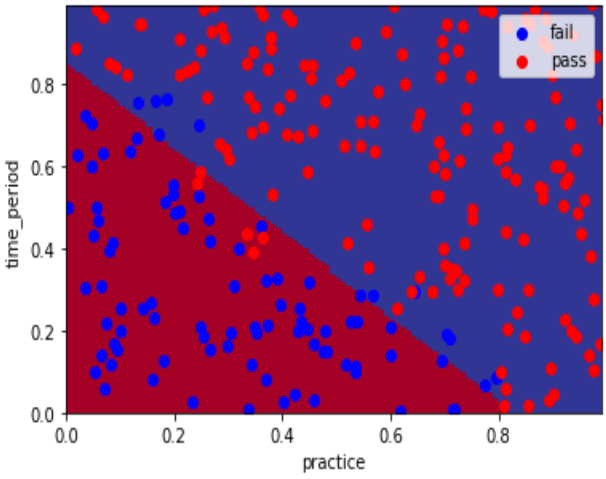
SVM（support Vector Mac）又称为支持向量机，是一种二分类的模型，是当下数学方法和最优化技术在机器学习分类中的最典型、最直接的应用。支持向量机可以分为线性核非线性两大类。其

主要思想：为找到空间中的一个更够将所有数据样本划开的超平面，并且使得本本集中所有数据到这个超平面的距离最短。就像我们可以用刀（二维平面）将西瓜（三维物体）分成两半，任何一个 n 维物体（空间）都可以被一个 n-1 维的超平面分成两部分。

如图所示，红点和蓝点分别代表两类样本，如果一条直线可以让两类样本中的点到这条直线的最短距离取最大值，一般认为这条直线就是最稳定的分界线，而决定这条直线的点往往是由少数几个支撑点决定的，这些点成为支持向量。



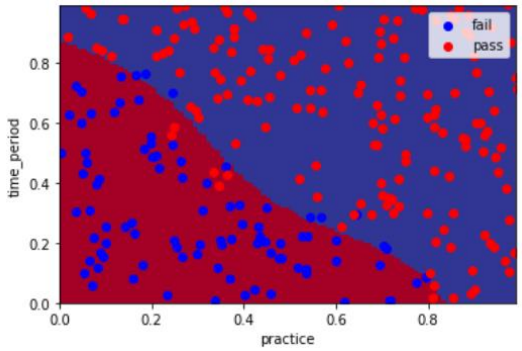
以二维的坐标点为例，支持向量算法就是要找一条直线将两类坐标点分开，而这种分割直线是由无数条的，但是在这些支线中，如果距离坐标点太近，那么噪声的扰动将对分类结果产生较大的影响，因此可以定义，SVM 算法就是找到其中距离训练样本最远的那条直线，也称作最优直线。



为了验证本算法，本报告手动编写函数 produce_data，生成 1000 个带了标签的二维坐标样本，样本的标签分别是 0,1，随后利用 sklearn.model_selection 中的 train_test_split，把 test_size 的参数设定为 0.25，则可以把前面的 1000 个二维坐标的 250 个作为测试样本，剩下的 750 个作为训练样本，把 750 个训练样本保存成 DataFrame 并以“SVM 算法训练样本.csv”保存到本地。

有了 750 个带有标签的训练样本，接下来可以直接用 sklearn.svm 库中的 SVM 实现 SVM 分类算法实验了，把之前定义好的 features_train 和 label_train，即训练样本的特征和标签，输入到 clf_SV2.fit 中，即可使用 SVM 分类算法进行分类，利用之前定义好的决策边界可视化函数 plot_pic，可以把决策边界画出来。

```
from sklearn.svm import SVC

clf_SVC = SVC()                                  #默认的核函数就是rbf;
clf_SVC.fit(features_train, label_train)         #训练;
pred_SVC = clf_SVC.predict(features_test)        #返回是测试集的预测的label;

plot_pic(clf_SVC, features_test, label_test)
```



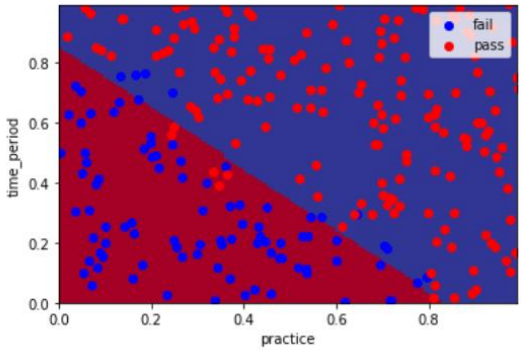
在 750 个样本使用 SVM 分类算法训练完成后，可以使用 clf_SVC.predict 对剩余的 250 个进行标签预测，返回的 pred_SVC2 代表 250 个测试样本的预测标签标签，再导入 sklearn.metrics 中的 accuracy_score 可以对 250 个测样样本的实际标签和预测标签进行评价。

```
from sklearn.metrics import accuracy_score
acc = accuracy_score(pred_SVC2, label_test)
acc
```

```
0.924
```

在 SVM 算法中，我们可以设定设置核函数类型，默认取值为‘rbf’，得出的曲线如上图所示并非完全直线，当我们设定 kernel = 'linear'时，可以得到一条直的决策边界线。

```
from sklearn.svm import SVC

clf_SVC2 = SVC(kernel = 'linear')
clf_SVC2.fit(features_train, label_train)
pred_SVC2 = clf_SVC2.predict(features_test)

plot_pic(clf_SVC2, features_test, label_test)
```



5.2 SVM 分类算法应用（自动给新评论分类）

与二维坐标类似的，之前爬取的每条评论都已经处理成了一个向量，代表空间的一个点，利用之前的得到的 TF-IDF 评论矩阵，再加上现在人工对爬取的 200 条评论进行分类标签（分别为 0,1）以此作为训练样本做 SVM 分类算法实验。

```
from sklearn.svm import SVC

clf_SVC = SVC()
clf_SVC.fit(pd.DataFrame(np.round(comment_vectorization_tv_ndarray, 2)),pd.Series(labels_true))
```

```
SVC()
```

对于每一条新的评论，之前已经讨论过，Scikit-Learn API 提供了 transform 函数，该函数可以根据我们之前提供的训练样本，当我们输入一条新评论时，该函数可以直接返回该新评论的特征向量，有了新评论的特征向量，可以以此为测试样本，让 SVM 聚类算法自动给新评论分类。

```
new_doc = '香港 祖国 红色 五星红旗 团结 感人 苦了 中华儿女 2008 陈凯歌 徐峥 黄渤 男孩 奥运 女排'
```

```
new_comment_vectorization_tv_df = pd.DataFrame(np.round(statistics_tv.transform([new_doc]).toarray(), 2),
             columns=statistics_tv.get_feature_names())
new_comment_vectorization_tv_df.to_csv('新评论特征向量提取.csv',encoding="utf_8_sig")
new_comment_vectorization_tv_df
```

	00	08	10	15	155	16	20	2008	2009	21	...	黄渤	黄渤宋佳	黄色	黑色幽默	默默无闻	黯淡无光	鼓掌	鼻子	齐聚	齐飞
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.37	0.0	0.0	...	0.32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

1 rows × 2700 columns

```
pred_SVC = clf_SVC.predict(new_comment_vectorization_tv_df)      #返回是测试集的预测的label;
```

```
pred_SVC
```

```
array([1], dtype=int64)
```