
目录

| | | |
|-------|---------------------------------|----|
| 1. | 环境配置说明..... | 2 |
| 2. | 数据爬取..... | 2 |
| 2.1 | 网页分析..... | 2 |
| 2.2 | Cookie 获取 | 2 |
| 2.3 | 获得微博具体 URL..... | 3 |
| 2.4 | 获得微博数据..... | 4 |
| 3. | 数据处理..... | 5 |
| 3.1 | 数据清洗..... | 5 |
| 3.2 | TF-IDF 加权与词袋模型 | 7 |
| 3.3 | 基于 SVD 分解的 PCA（主成分分析）数据降维 | 9 |
| 4. | 数据分析与可视化..... | 10 |
| 4.1 | 数据聚类..... | 10 |
| 4.2 | 数据可视化..... | 11 |
| 4.2.1 | 热度变化趋势..... | 11 |
| 4.2.2 | 词云..... | 12 |
| 4.2.3 | 谣言转发和点赞之间的关系..... | 13 |
| 4.3 | 数据分类..... | 14 |
| 5. | 不足与可能的改进..... | 16 |
| 5.1 | 数据存储方式..... | 16 |
| 5.2 | 数据源的改进..... | 16 |

1. 环境配置说明

- Python 解释器版本: Python 3.9 Anaconda
- 操作系统: Windows 10
- IDE: PyCharm 2021.3 (Professional Edition)
- 第三方 Python Packages: [selenium](#), [BeautifulSoup\(bs4\)](#), [pandas](#), [jieba](#), [numpy](#), [sklearn](#), [matplotlib](#), [wordcloud](#)

Python 额外程序包可以使用 [pip/conda install](#) 在 IDE 终端直接安装。或者也可以使用 PyCharm 的包安装程序（见图 1）。

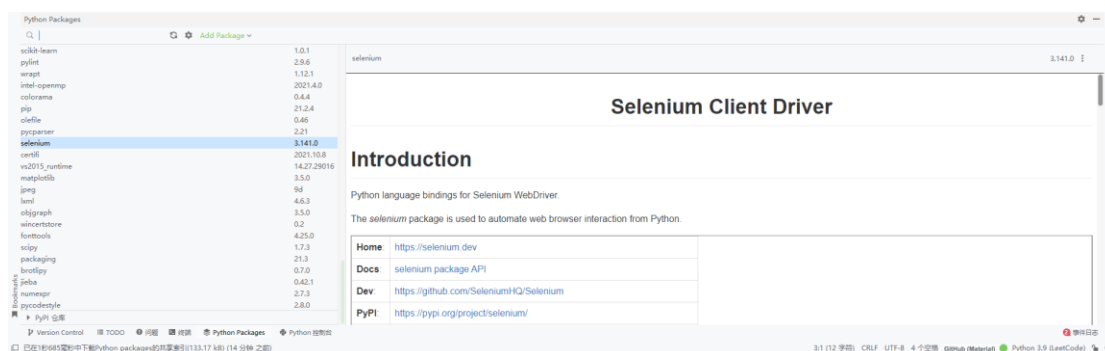


图 1PyCharm 的包安装程序

注：本项目数据处理的最终结果是文件 `analysis/CSV/weibo_data_classified.csv`

2. 数据爬取

2.1 网页分析

微博的所有网页均采用了 js 动态渲染技术，即使采用正确的 Cookie，利用 [requests](#) 等处理静态网页的库爬取网页仅能得到一大串无法使用的 Java Script 脚本。因此我最终决定采用 [selenium](#) 库，配合 [BeautifulSoup](#)，利用浏览器驱动模拟真实浏览器获得 Cookie、发出请求。

2.2 Cookie 获取

本节代码: `spider/GetCookie.py`

通过多次尝试，我发现，微博网页的 Cookie 获取无论是通过账号密码或是邮箱方式，最后都必须通过手机短信或者手机扫码才能成功，即完全自动化的 Cookie 获取是比较困难的。因此我利用 [selenium](#) 来自动跳转至扫码界面（图 2）：

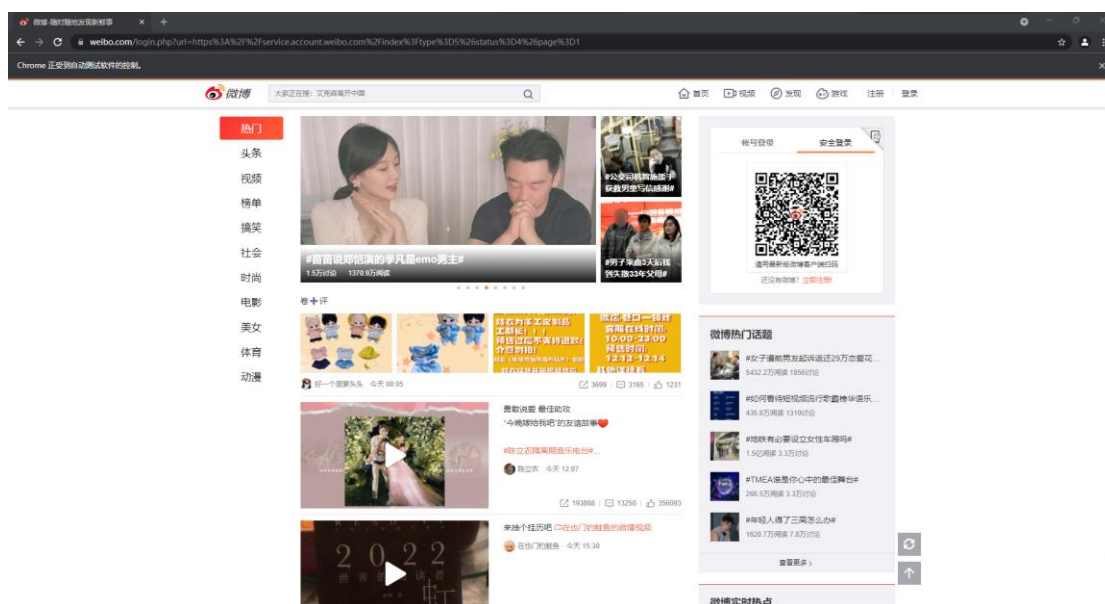


图 2 自动跳转至该页面

随后，程序将有 30 秒的时间让我通过手机扫码登录。经过 30 秒并登录成功后，程序将提取 `selenium` 所模拟的浏览器中取得的数据，即我所需的 Cookie。最后将其以 txt 形式保存至本地：`spider/weibo_cookies.txt`。

```

6      driver = webdriver.Chrome()
7      driver.maximize_window()
8      driver.get('https://service.account.weibo.com/index?type=5&status=4&page=1')
9      sleep(6)
10     # 模拟点击手机扫码登录
11     driver.find_element_by_xpath('//*[@id="pl_login_form"]/div/div[1]/div/a[2]').click()
12     # 在30秒的时间内扫码登录，等待程序提示后cookie就保存到本地了
13     sleep(30)
14     dictCookies = driver.get_cookies() # 获取cookies

```

图 3 模拟登录主要代码段

2.3 获得微博具体 URL

本节代码：`spider/GetInfoUrl.py`

获得 Cookie 后，我发现如果直接遍历谣言微博页面并一条条处理微博内容将极为耗时，因为需要获得微博的转评赞、博主是否是 VIP 等具体数据，这需要进入谣言微博原文获取。然而谣言微博是不一定有原文信息的，有的谣言微博甚至内容已经完全删除。因此我决定首先获取每条谣言微博信息的 URL 并保存到本地：`spider/info_urls.txt`。

2.4 获得微博数据

本节代码: `spider/GetWeiboContent.py`

首先初始化 `selenium` 浏览器驱动, 并获取 2.3 节中得到的谣言微博信息 `url`。遍历这些 `url`, 并且提取其具体微博特征(包括: 内容、发布时间、转发量、评论量、点赞量、博主是否 VIP、是否含有链接)。最后得到的数据利用 `pandas` 处理并存储到 CSV 中: `spider/CSV/weibo_data.csv`。由于数据量较大, 我采用了边处理边 IO 的方案, 即处理 100 条微博就将数据存入 CSV 中, 最后获取的微博数据量为 **7037 条**。

```
57 for single_url in target_urls:
58
59     print('共条' + str(len(target_urls)) + '微博, 正在处理第' + str(count) + '条')
60     count += 1
61
62     # 每隔100条微博保存一次
63     if count == 100:
64         print('正在写入csv...')
65         df.to_csv('CSV/weibo_data.csv')
66         print('csv写入完成!')
67         df = pd.DataFrame(columns=['content', 'time', 'forward', 'comment', 'like', 'VIP', 'link'])
68     elif count != 0 and count % 100 == 0:
69         print('正在写入csv...')
70         df.to_csv('CSV/weibo_data.csv', mode='a', header=False)
71         print('csv写入完成!')
72         df = pd.DataFrame(columns=['content', 'time', 'forward', 'comment', 'like', 'VIP', 'link'])
73
74     driver.get(single_url.strip('\n'))
75     time.sleep(0.5)
76     soup = BeautifulSoup(driver.page_source, 'lxml')
77     weibo_data = {'content': '', 'time': '', 'forward': 0, 'comment': 0, 'like': 0, 'VIP': 0, 'link': 0}
78
79     # 如果没有原文信息, 那么只能获得微博的基本信息
80     # 首先判断是否是否, 其次看网页中被告人的微博的描述是否存在(有被屏蔽和已删除两种可能), 可以通过是否能点击被告人的ID链接来判断
81     if soup.find('div', class_='result win') is not None and \
82        soup.find('div', class_='feed bg-orange2 clearfix').find('a') is not None:
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109     # 找到脏话且有原文的微博
110     if soup.find('div', class_='result win') is not None and soup.find('a', attrs={
111         'suda-utrack': 'key=tblog_service_account&value=original_text'}) is not None:
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
```

3. 数据处理

3.1 数据清洗

本节代码: *processing/DataCleaning.py*

数据清洗方面，主要采用 `jieba` 库进行处理。因为绝大多数中文文本的特征集中在名词与动词上（特别是名词），因此数据清洗思路为：首先提取微博内容的中文部分，然后提取微博内容的以下特征：

1. 仅包含名词的文本
2. 仅包含动词的文本
3. 仅包含名词的**去重**文本
4. 仅包含动词的**去重**文本
5. 人名
6. 地名
7. 机构名
8. **是否疫情相关**（通过疫情相关词语表 *processing/covid_19_word.txt* 判断）

清洗后的数据需要满足以下需求：

- 删除不包含中文的微博
- 严格按照时间顺序（精确到分钟）排序
- 对于一个事件，去除对其重复的谣言微博

首先利用正则表达式取得微博的中文部分，其中也间接获得了微博中的**中文字数**这一特征值

```

73 def chinese_word_only(s: str):
74     """
75     将字符串处理为仅保留中文字符
76     :param s: 目标字符串
77     :return: 一个2个元素的列表，第一个元素是仅中文字符串，第二个元素是其长度
78     """
79     pattern = re.compile(r'^[\u4e00-\u9fa5]*$')
80     res = re.sub(pattern, '', s)
81     return [res, len(res)]

```

图 6 取得微博的中文部分

利用分词，获得微博所有的所需特征

```

31 def get_noun_word(s: str):
32     """
33     利用jieba分词，提取微博内容中的名词和动词
34     :param s: 微博内容
35     :return: 仅包含名词的文本(空格间隔，下同)，仅包含动词的文本，仅包含名词的去重文本，人名，地名，机构名，是否疫情相关
36     """
37     res_n = []
38     res_v = []
39     res_per = []
40     res_loc = []
41     res_org = []
42     covid19_correlation = 0
43     words = ps.cut(s, use_paddle=True)
44     for word, flag in words:
45         # print('%s %s' % (word, flag))
46         # 若为名词且不在停用词表中，则加入写入串
47         if not is_stop_word(word):
48             res_n.append(word)
49             res_v.append(word)
50             res_per.append(word)
51             res_loc.append(word)
52             res_org.append(word)
53             covid19_correlation = 1 if word in ['新型冠状病毒', '武汉肺炎', '新冠肺炎', '2019新型冠状病毒'] else 0
54     return [' '.join(res_n), ' '.join(res_v), ' '.join(list(set(res_n))), ' '.join(list(set(res_v))),
55           ' '.join(list(set(res_per))), ' '.join(list(set(res_loc))), ' '.join(list(set(res_org))),
56           covid19_correlation]

```

图 7 获取微博文本特征

较为困难的部分为判断相同内容的谣言微博。观察原始数据，我发现根据爬虫的结果，谣言微博内容是根据时间顺序排列的（原始数据的精度为日），并且时间跨度较大。因此我采用回溯法，根据在 CSV/pandas.DataFrame 中的行号，如果回溯 10 行后发现至少存在一个地名相同，或者至少存在两个普通名词相同（请注意，在 jieba 分词的结果中，普通名词并不包含地名），那么就可以判断这条微博和已有微博重复了，在微博数据特征中添加一条 `need_del` 并赋值为 1 以便后续清除。

```

80 def judge_equal_content(df_data: pd.DataFrame, index: int, jieba_result: list):
81     """
82     判断微博内容是否相同，通过其中包含的相同名词和地名来判断，并且只回溯相邻的10条微博
83     :param jieba_result: jieba分词结果
84     :param df_data: 数据矩阵
85     :param index: 欲判断的数据行号(在CSV/DataFrame中的行号)
86     :return: 存在相同内容微博则返回1，否则返回0
87     """
88     set_base_n = set(jieba_result[0].split())
89     set_base_loc = set(jieba_result[5].split())
90     start_index = 0
91     if index > 10:
92         start_index = index - 10
93     for i in range(start_index, index):
94         if isinstance(df_data.loc[i, 'LOC'], str):
95             set_test = set(df_data.loc[i, 'LOC'].split())
96             # 如果至少存在一个地名相同
97             if len(set_base_loc & set_test) > 0:
98                 return 1
99         if isinstance(df_data.loc[i, 'content_n'], str):
100             set_test = set(df_data.loc[i, 'content_n'].split())
101             # 如果至少存在两个名词相同
102             if len(set_base_n & set_test) > 1:
103                 return 1
104     return 0

```

图 8 判断是否是需要被删除的重复内容微博

最后，综合利用以上所有方法对数据进行清洗，并按照时间进行降序排序（精度为分），输出结果至 `processing/CSV/weibo_data_clean.csv`。清洗后包含 2243 条微博数据。

```

107 if __name__ == '__main__':
108     columns_list = ['content', 'content_n', 'content_v', 'PER', 'LOC', 'ORG', 'content_n_set', 'content_v_set', 'time',
109                    'forward', 'comment', 'Like', 'VIP', 'link', 'words', 'covid19', 'need_del']
110     print(jieba.__version__)
111     jieba.enable_paddle() # 启动paddle模式，提高精度，但会增加处理时间
112     df = pd.read_csv('../spider/CSV/weibo_data.csv', index_col=None, usecols=[i for i in range(1, 8)])
113     df = df.reindex(columns=columns_list)
114     for i in range(len(df)):
115         # 调用图7和图8的方法
116         res = get_noun_word(df.loc[i, 'content'])
117         df.loc[i, 'content_n'] = res[0]
118         df.loc[i, 'content_v'] = res[1]
119         df.loc[i, 'content_n_set'] = res[2]
120         df.loc[i, 'content_v_set'] = res[3]
121         df.loc[i, 'PER'] = res[4]
122         df.loc[i, 'LOC'] = res[5]
123         df.loc[i, 'ORG'] = res[6]
124         df.loc[i, 'covid19'] = res[7]
125         df.loc[i, 'need_del'] = judge_equal_content(df, i, res)
126     df.drop(df[(df['words'].isnull() | (df['need_del'] == 1)).index], inplace=True) # 删除不合格的数据
127     df.drop_duplicates(subset=['content_v_set'], inplace=True) # 删除内容完全重复的数据
128     df['time'] = pd.to_datetime(df['time']) # 设置时间格式，按照时间排序
129     df.sort_values("time", inplace=True, ascending=False)
130     print(df)
131     df.to_csv('CSV/weibo_data_clean.csv', index=False)

```

图 9 调用所有相关方法进行数据清洗

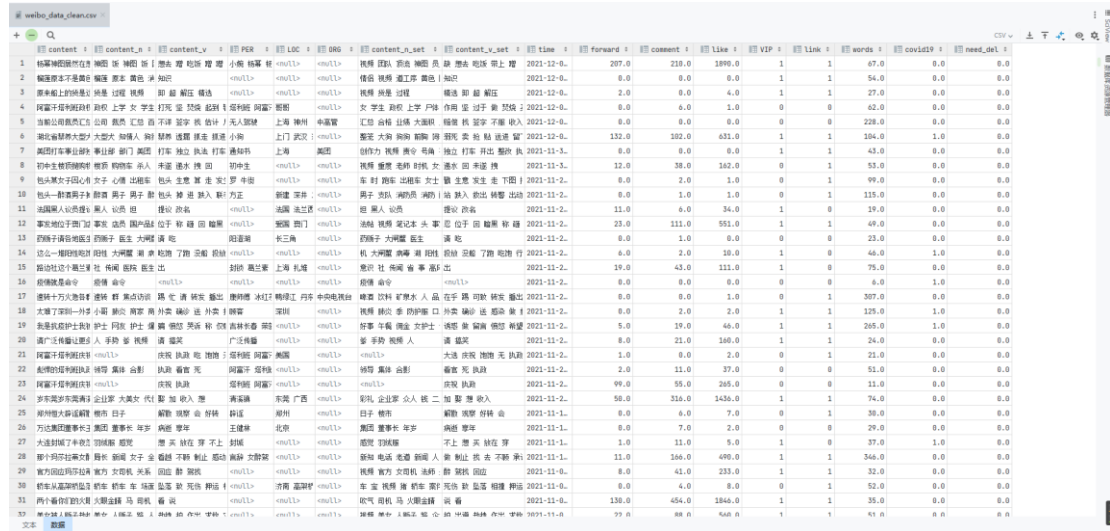


图 10 清洗后的微博数据

3.2 TF-IDF 加权与词袋模型

本节代码: `processing/GetEigenvector.py`

建立词袋模型, 首先需要分析词频数据, 在 3.1 节的基础上, 将各类名词和动词结合并分析。通过 Python 的 `collections` 库很容易就能取得词频信息, 并将其存储至 CSV 中: `processing/CSV/tf_vector.csv`

```
20 # 方法1, 传统词频
21 counter_obj = Counter(v.split())
22 v_dict = dict(counter_obj)
23 # print(counter_obj)
24 df_res = pd.DataFrame(v_dict, index=['num']).T
25 print(df_res.columns)
26 df_res.sort_values("num", inplace=True, ascending=False)
27 print(df_res.info())
28 df_res.to_csv('CSV/tf_vector.csv') # 该csv表示所有数据的词频向量
```

图 11 获得词频矩阵

显然, 普通的词频矩阵描述数据特征是不够准确的, 采用 TF-IDF 加权后的词频数据才更具代表性。我决定采用 `jieba` 库的 TF-IDF 加权, 分析得出权重最高的 1000 个词, 并存储: `processing/CSV/tf_idf_weight.csv`

```
30 # 方法2, jieba的tf-idf分析, 找到权重最大的前1000个词
31 tags = jieba.analyse.extract_tags(v, topK=1000, withWeight=True)
32 tags_dic = dict(tags)
33 # for tag, value in tags_dic.items():
34 #     print("tag: %s\t\t weight: %f" % (tag, value))
35 df_key_words = pd.DataFrame({'tag': tags_dic.keys(), 'weight': tags_dic.values()})
36 print(df_key_words.info())
37 df_key_words.to_csv('CSV/tf_idf_weight.csv', index=False)
```

图 12TF-IDF 加权

观察两种分析方式的结果可以看出相对传统词频数据，**jieba** 的 TF-IDF 加权去除了许多没有代表性或是过短的名词，如人、事等，数据更有代表性

| tag | weight |
|------|------------------------|
| 视频 | 0.17491270452531102 |
| 转发 | 0.060880057585078177 |
| 孩子 | 0.053214056527812185 |
| 阿四 | 0.039565696717009005 |
| 死亡 | 0.03846917347283294 |
| 链接 | 0.036636140407469014 |
| 肉松 | 0.03352359092727776 |
| 乔任 | 0.0312461252036069 |
| 学生 | 0.024987630666053317 |
| 发生 | 0.023803476117364164 |
| 感染 | 0.02243012235372375 |
| 警察 | 0.022131728480297343 |
| 医院 | 0.0205858248708219 |
| 警察 | 0.020387141661483966 |
| 小孩 | 0.019928345160982748 |
| 医院 | 0.019289715904086282 |
| 男子 | 0.01926275410988592 |
| 女子 | 0.0188922935932376 |
| 病毒 | 0.017790673209328128 |
| 隔离 | 0.017593803797770674 |
| 现场 | 0.017469077141465512 |
| 新闻 | 0.01718642433637096 |
| 朋友 | 0.017080511234888227 |
| 希望 | 0.01664632639275299 |
| 家长 | 0.016528948341008886 |
| 儿童 | 0.015900515875418342 |
| 网友 | 0.0152599853120599 |
| 爆料 | 0.015049575366602194 |
| 电视新闻 | 0.014988355729021248 |
| 抢救 | 0.014964578397275915 |
| 事件 | 0.014713183962415671 |
| 甲女生 | 0.01464777616459397606 |

| <anonymous> | num |
|-------------|-----|
| 人 | 783 |
| 视频 | 703 |
| 孩子 | 334 |
| 吃 | 295 |
| 清 | 230 |
| 拍 | 212 |
| 转发 | 208 |
| 番 | 207 |
| 死亡 | 200 |
| 说 | 184 |
| 发生 | 178 |
| 会 | 158 |
| 中国 | 150 |
| 典 | 149 |
| 去 | 145 |
| 链接 | 137 |
| 学生 | 137 |
| 死 | 135 |
| 阿四 | 135 |
| 秒 | 112 |
| 感染 | 107 |
| 医院 | 106 |
| 希望 | 104 |
| 朋友 | 95 |
| 事 | 94 |
| 女子 | 94 |
| 肉松 | 93 |
| 新闻 | 92 |
| 男子 | 91 |
| 现场 | 89 |
| 警察 | 88 |
| 小孩 | 86 |

图 13 两种词袋模型构建方法对比

在 TF-IDF 加权的基础上，构建词袋模型，生成 2243×1000 的词袋矩阵：`processing/CSV/bw_matrix.csv`

```

40     bw_matrix = np.zeros(shape=(len(df_weibo_data), len(df_key_words)))
41     key_words = list(tags_dic.keys())
42     for i in range(len(df_weibo_data)):
43         # 遍历所有文章
44         word_list = df_weibo_data.loc[i, 'data'].split() # 取得文章内容
45         for w in word_list:
46             # 判断文章每个词是否在1000个关键字里
47             if w not in key_words:
48                 continue
49             else:
50                 # 若为关键字，则矩阵相应位置加一
51                 bw_matrix[i][key_words.index(w)] += 1
52
53     bw_matrix_df = pd.DataFrame(data=bw_matrix, columns=key_words)
54     bw_matrix_df.to_csv('CSV/bw_matrix.csv', index=False) # 该csv即基于TF-IDF加权的词袋模型矩阵
55 
```

图 14 生成词袋矩阵

图 15 生成的矩阵大小为 2243×1000

3.3 基于 SVD 分解的 PCA（主成分分析）数据降维

本节代码: `processing/PCA.py`

调用 `sklearn` 库的相关方法对 TF-IDF 词袋矩阵进行降维。利用基于 SVD 分解的主成分分析，分析结果表明，如果需要保留特征值 97% 的信息量，那么特征值维度最多可以从 1000 降低至 534。则表示只需要分析前 534 个 TF-IDF 关键词，就能代表 1000 个 TF-IDF 关键词 97% 的信息量。

```

1 import pandas as pd
2 from sklearn.decomposition import PCA
3
4 # 利用基于SVD分解的PCA（主成分分析）进行数据降维
5
6 if __name__ == '__main__':
7     data = pd.read_csv('CSV/bw_matrix.csv').values
8     # svd_solver = 'full' 该参数是指定SVD的计算方式的
9     # 表示希望降维后的总解释性方差占比大于n_components指定的百分比，即说，希望保留百分之多少的信息量。
10    pca = PCA(n_components=0.97, svd_solver='full')
11    pca.fit(data)
12
13    print('降维后的特征向量所保留信息量的百分比: ' + str(sum(pca.explained_variance_ratio_))) # 0.9701178185590625
14    print('特征维度从1000降低至: ' + str(len(pca.explained_variance_ratio_))) # 534
15
Python 控制台 × PCA (1) ×
D:\app\anaconda\envs\LeetCode\python.exe "D:\app\PyCharm 2020.1.3\plugins\python\helpers\pydev\pydevconsole.py" --mode=client --port=9281
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['D:\\code\\ENPythonProject', 'D:\\code\\ENPythonProject'])
Python 控制台
降维后的特征向量所保留信息量的百分比: 0.9701178185590625
特征维度从1000降低至: 534
>>>

```

图 16 程序内容与执行结果

4. 数据分析与可视化

4.1 数据聚类

本节代码: *analysis/LDA.py*

对处理后的数据通过基于 `sklearn` 的 LDA 主题模型进行分类。首先需要将词袋模型转换为 `sklearn` 库的 `CountVectorizer` 数据结构, 然后进行 LDA 主题模型的训练。主要参数有两个, 一个是 `n_components`, 代表需要分出的类别数量; 另一个是 `max_iter`, 代表模型训练的最大迭代次数。在经过多次调试后, 我发现 `n_components` 设置为 3, `max_iter` 设置为 5 是结果较好的参数值。

```

11 df_data = pd.read_csv('../processing/CSV/weibo_data_all_words.csv')
12 cntVector = CountVectorizer()
13 # 为了配合使用sklearn库, 此处采用sklearn提供的词向量词袋模型
14 cntTf = cntVector.fit_transform(df_data['data'])
15 print('Vector done!')
16 class_num = 3 # 最关键的参数, 该参数即分类数
17 # max_iter : EM算法的最大迭代次数。
18 # learning_method: 即LDA的求解算法。有 'batch' 和 'online'两种选择。前者较为简单快速, 后者较为复杂、参数较多。
19 lda = LatentDirichletAllocation(n_components=class_num, max_iter=5, learning_method='batch', random_state=0)
20 doc = lda.fit_transform(cntTf)
21 print(doc) # 文档的主题模型分布在doc中, 即表示每篇文档属于哪一类
22 # print(len(doc))
23 print('-' * 30)
24 print(lda.components_) # 主题词分布在lda.components_中
25 # print(len(lda.components_))
26 # print(len(lda.components_[0]))

```

图 17LDA 训练

Python 控制台

```

Vector done!
[[0.95405361 0.02299669 0.0229497 ]
 [0.03224994 0.03341581 0.93433424]
 [0.88536326 0.05725492 0.05738183]
 ...
 [0.02458653 0.95145123 0.02396224]
 [0.01851587 0.96111517 0.02036895]
 [0.01575092 0.96655097 0.01769811]]

-----
[[0.33343452 1.29228778 0.33358201 ... 1.33309406 0.3336378 0.33350847]
 [0.33343305 0.37317556 0.33354242 ... 0.33351029 0.3335752 0.33348311]
 [1.33313243 0.33453666 2.33287557 ... 0.33339565 1.332787 1.33300842]]

```

图 18 训练结果

随后将分好的不同微博数据分别写入不同的 CSV 之中，并也在源数据的基础上增加特征值 *class*。文件均保存于 *analysis/CSV* 文件夹中。

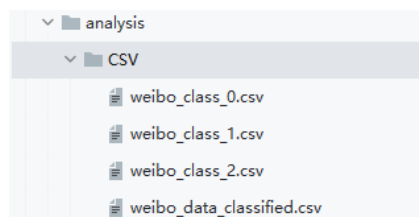


图 19 分类数据保存位置

对于分类结果，通过基本的语义分析可以得出以下结论：

- **第一类**为社会类，谣言包括时政、国际、警匪冲突等方面；
- **第二类**为生活类，谣言包括平时生活中的小知识和娱乐明星等；
- **第三类**为事故类，明显的特征是该类谣言绝大多数都和人的生命安全有关。

这些结果在下文的可视化-词云部分能够进一步印证。

4.2 数据可视化

4.2.1 热度变化趋势

本节代码：*analysis/VisualizationTrend.py*

在本节中，我将展示所有类别的不实信息随时间变化（2016-2021）的热度，以年为单位展示各个类别微博数据变化量。

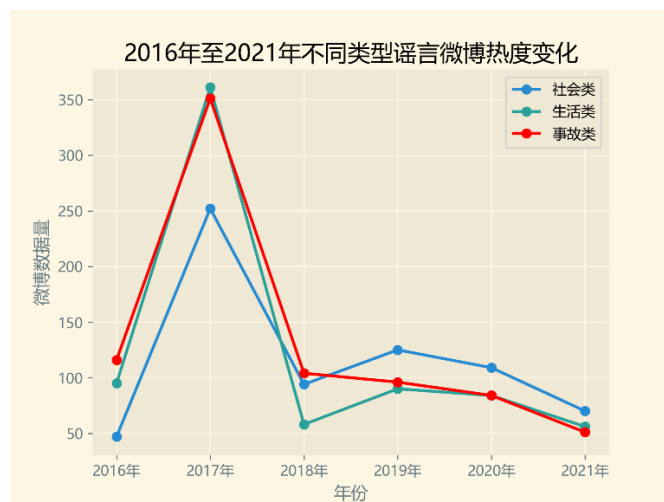
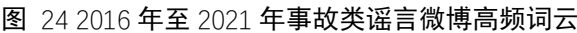
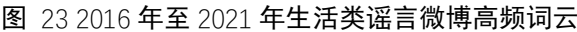


图 20 2016 年至 2021 年不同类型谣言微博热度变化

如图 20 所示，各个类别谣言热度变化曲线类似。在 2018 年以前，生活类和事故类谣言微博数据量占据优势。然而 2018 年至今，社会类谣言微博则在热度上领先其它两类。

需要注意的是 2017 年所有类别谣言微博数据量的激增，这可能与 2017 年微博打击谣言政策力度收紧有很大关联。



4.2.3 谣言转发和点赞之间的关系

针对谣言微博，本节将研究其转发与点赞量之间的关系。首先利用 `pandas` 剔除极端数据，再绘制散点图与线性拟合曲线，结果如图 25 所示。

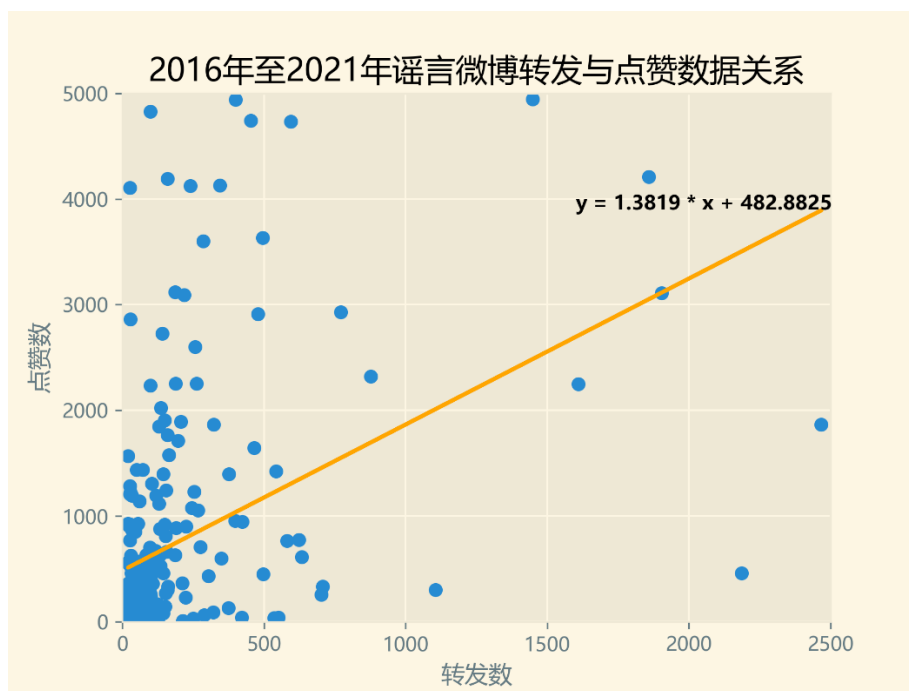


图 25 2016 年至 2021 年谣言微博转发与点赞数据关系

可以看到点赞数是随着转发数的增长而增长的。然而我们还应该注意到，有很大一部分微博是仅仅有点赞而无转发数的，这和人们比起分享消息更喜欢简单地对微博点赞的普遍习惯有关，同时也不排除有用户怀疑谣言微博内容从而仅点赞不转发的情况。

而从数据量来看，谣言微博保持相对较低的点赞和转发数据量，这很可能是其在传播过程中被微博反谣言系统监测并限制流量，这也能说明微博的反谣言措施是有一定效果的。

4.3 数据分类

本节代码: *analysis/SVM.py*

本节将使用基于 `sklearn` 的 SVM 分类模型训练判断是否为谣言微博的分类器。在本节，我使用了 3.3 节的结论，仅采用 534 个特征向量参与训练。对于训练集，选取前 2000 条谣言微博和额外随机爬取的 1000 条正常微博。对于测试集，选取剩余的 243 条谣言微博与另外不同的 500 条正常微博。即训练集 3000 条数据，测试集 743 条数据。对于 SVM 模型，经过多次调试发现，采用 `rbf` 核函数，参数 `C` 取 6，`gamma` 取 0.001 得到的结果最好。按如图 26 所示代码进行训练：

```

13 # 根据PCA降维，仅需要534特征词即可代表绝大多数数据特征
14 df_fake = pd.read_csv('../processing/CSV/bw_matrix.csv', usecols=[i for i in range(534)])
15 df_normal = pd.read_csv('../processing/CSV/bw_matrix_normal.csv', usecols=[i for i in range(534)])
16 # 前2000条虚假微博和前1000条正常微博进行训练
17 df_train = pd.concat([df_fake.loc[:1999], df_normal.loc[:999]], axis=0, ignore_index=True)
18 # 剩余的所有虚假微博和另外的500条正常微博进行测试
19 df_test = pd.concat([df_fake.loc[2000:], df_normal[1000:1500]], axis=0, ignore_index=True)
20 print(df_train.info())
21 # 2000个虚假信息，1000个真实信息
22 class_arr = np.array([0 for i in range(2000)] + [1 for i in range(1000)])
23 print(class_arr)
24
25 model = SVC(kernel='rbf', C=6, gamma=0.001)
26 start = time.time()
27 model.fit(df_train, class_arr)
28 end = time.time()
29 print('Train time: %s Seconds' % (end - start))
30 start = time.time()
31 pre = model.predict(df_test)
32 end = time.time()
33 print('Test time: %s Seconds' % (end - start))
34 # print(len(pre))
35 # print(pre)

```

图 26 数据处理与训练

```

37 # 测试集信息
38 right_arr = np.array([0 for i in range(len(df_fake.loc[2000:]))] + [1 for i in range(len(df_normal[1000:1500]))])
39
40 # 计算准确率 (accuracy)
41 accuracy = metrics.accuracy_score(right_arr, pre)
42 print("准确率为: \n", accuracy)
43 # 计算精确率 (precision)
44 precision = metrics.precision_score(right_arr, pre, average=None)
45 print("精确率为: \n", precision)
46 print('均值{:.4f}\n'.format(sum(precision) / 10))
47 # 计算召回率 (recall)
48 recall = metrics.recall_score(right_arr, pre, average=None)
49 print("召回率为: \n", recall)
50 print('均值{:.4f}\n'.format(sum(recall) / 10))
51 # 计算F1-score (F1-score)
52 F1_score = metrics.f1_score(right_arr, pre, average=None)
53 print("F1值为: \n", F1_score)
54
55 cp = metrics.classification_report(right_arr, pre)
56 print("-" * 25 + "分类报告" + "-" * 25 + "\n", cp)
57
58 if __name__ == '__main__':
59     Python 控制台 x SVM
60     -----分类报告-----
61
62         precision    recall  f1-score   support
63
64         0           1.00      0.84      0.92         243
65         1           0.93      1.00      0.96         500
66
67         accuracy          0.95          743
68         macro avg       0.96      0.92      0.94          743
69         weighted avg     0.95      0.95      0.95          743
70
71 >>>

```

图 27 输出的分类报告

根据图 27 的分类报告我们可以得知，虚假微博的精确率（precision）为 1，说明模型没有将真实微博判断为虚假微博的情况。而真实微博的召回率（recall）为 1，说明了同样的情况，即测试集中没有正常微博被认为是谣言微博。谣言微博的找回率为 0.84，这说明了存在 0.16 左右的概率，模型将谣言微博误认为正常微博。模型总体准确率为 0.95，是非常不错的数据结果。

5. 不足与可能的改进

5.1 数据存储方式

在第 3 节中，我尝试生成关键词-微博矩阵以表示词袋模型，很显然，这个矩阵是相对较大的一个稀疏矩阵。在数据持久化部分，除了 `pandas` 以外或许可以考虑 `scipy` 库中的稀疏矩阵数据结构对其进行存储，这样节省了空间也能在一定程度上提高数据处理速度。

5.2 数据源的改进

在 4.3 节中，我们看到基于 SVM 模型中正常微博的召回率为 1，这显然不是非常实际的。我认为出现这个结果的一个原因在于数据量过小，另一个原因在于正常微博和取得的谣言微博之间的差距过大。通过 4.1 节我们可以看到，谣言微博分为不同类别是可能且相对简单的，然而通过随机爬取的正常微博则复杂的多。举例来说，一位用户在微博上分享他今天的晚餐，或是在微博上发送图片，这些微博很显然不好分类，而且明显能够看出其相对谣言微博之间的区别。如果在微博数据上的处理粒度更小，那么就有可能得到更加真实的结果。