


```
pythonProject - fenci.py
Project
  pythonProject
    chuli.py
    chulicixing.py
    cixing_tongji.py
    CoreNatureDictionary.csv
    cws.csv
    danju.csv
    fenci.py
    main.py
    my_cws_corpus.csv
    renmin.csv
    renmin.txt
    renmincixing.csv
    renmincixing.txt
    temp.csv
    test.csv
    tongjifenci.py
    train_lm
  External Libraries
  Scratches and Consoles
  Structure
  Favorites

# 正向最长匹配
# 从当前扫描位置的单词所有可能的结尾，我们找最长的
def forward_segment(text, dic):
    word_list = []
    i = 0
    while i < len(text):
        longest_word = text[i]
        for j in range(i + 1, len(text) + 1):
            word = text[i:j]
            if word in dic:
                if len(word) > len(longest_word):
                    longest_word = word
        word_list.append(longest_word)
        i += len(longest_word)
    return word_list

# 逆向最长匹配
def back_segment(text, dic):
    word_list = []
    i = len(text) - 1
    while i >= 0:
        longest_word = text[i]
        for j in range(0, i):
            word = text[j:i + 1]
            if word in dic:
                if len(word) > len(longest_word):
                    longest_word = word
        word_list.append(longest_word)
        i -= len(longest_word)
    word_list.reverse()
    return word_list

for i in range(test_sents_num) for tmp in ans_list
```

```
pythonProject - fenci.py
Project
  pythonProject
    chuli.py
    chulicixing.py
    cixing_tongji.py
    CoreNatureDictionary.csv
    cws.csv
    danju.csv
    fenci.py
    main.py
    my_cws_corpus.csv
    renmin.csv
    renmin.txt
    renmincixing.csv
    renmincixing.txt
    temp.csv
    test.csv
    tongjifenci.py
    train_lm
  External Libraries
  Scratches and Consoles
  Structure
  Favorites

longest_word = text[i]
for j in range(0, i):
    word = text[j:i + 1]
    if word in dic:
        if len(word) > len(longest_word):
            longest_word = word
word_list.append(longest_word)
i -= len(longest_word)
word_list.reverse()
return word_list

# 统计单字成词的个数
def count_single_char(word_list: list):
    return sum(1 for word in word_list if len(word) == 1)

# 双向最长匹配，实际上是做了个融合
def bidirectional_segment(text, dic):
    f = forward_segment(text, dic)
    b = back_segment(text, dic)
    # 词数更少优先
    if len(f) < len(b):
        return f
    elif len(f) > len(b):
        return b
    else:
        # 单字更少优先
        if count_single_char(f) < count_single_char(b):
            return f
        else: # 都相等时我们更倾向于逆向匹配的
            return b

dic = load_dictionary()
for i in range(test_sents_num) for tmp in ans_list
```

我们随意输入几个句子并输出结果，根据结果来看，各个方法分词的效果还算不错。接下来我们使用人民日报的分好的语料库进行一个全篇的预测。

```
xun_lian = forward_segment(test_sents[i], dic)
xun_lian_list = zhuan_huan(xun_lian)
ans_list = zhuan_huan(ans_sents[i])

xun_lian_set = set()
for tmp in xun_lian_list:
    xun_lian_set.add(tuple(tmp))
ans_list_set = set()
for tmp in ans_list:
    ans_list_set.add(tuple(tmp))
TP = ans_list_set & xun_lian_set
p = len(TP) / len(xun_lian_list)
r = len(TP) / len(ans_list)
P += p
R += r
if i % 100 == 0:
    print(i / test_sents_num)

# 求一个平均值
P = P / test_sents_num
R = R / test_sents_num
F1 = 2 * P * R / (P + R)
print("P,R,F1", P, R, F1)
end_time = time.time()
print(end_time - start_time)

for i in range(test_sents_num)
```

Run: P, R, F1 0.8412775520097979 0.8864272778728313 0.8632624710717806
26.91647696495056

Process finished with exit code 0

首先我们使用前向切分的方法，可以看到 P, R, F1 三个值，以及用时 26.92s。

```
i += len(word)
return ans

test_sents_num = len(test_sents)
print(test_sents_num)
P = 0
R = 0
for i in range(test_sents_num):
    xun_lian = back_segment(test_sents[i], dic)
    xun_lian_list = zhuan_huan(xun_lian)
    ans_list = zhuan_huan(ans_sents[i])

    xun_lian_set = set()
    for tmp in xun_lian_list:
        xun_lian_set.add(tuple(tmp))
    ans_list_set = set()
    for tmp in ans_list:
        ans_list_set.add(tuple(tmp))
    TP = ans_list_set & xun_lian_set
    p = len(TP) / len(xun_lian_list)
    r = len(TP) / len(ans_list)
    P += p
    R += r
    if i % 100 == 0:
        print(i / test_sents_num)

# 求一个平均值
***
```

Run: P, R, F1 0.8481106838880893 0.8934082673412537 0.8701751088698839
27.511301040649414

Process finished with exit code 0

接下来使用后向切分的分法，正确率微微提升，同时用时 27.51s。

```

pythonProject - fenci.py
113         i += len(word)
114         return ans
115
116
117 test_sents_num = len(test_sents)
118 print(test_sents_num)
119 P = 0
120 R = 0
121 for i in range(test_sents_num):
122     xun_lian = bidirectional_segment(test_sents[i], dic)
123     xun_lian_list = zhuan_huan(xun_lian)
124     ans_list = zhuan_huan(ans_sents[i])
125
126     xun_lian_set = set()
127     for tmp in xun_lian_list:
128         xun_lian_set.add(tuple(tmp))
129     ans_list_set = set()
130     for tmp in ans_list:
131         ans_list_set.add(tuple(tmp))
132     TP = ans_list_set & xun_lian_set
133     p = len(TP) / len(xun_lian_list)
134     r = len(TP) / len(ans_list)
135     P += p
136     R += r
137     if i % 100 == 0:
138         print(i / test_sents_num)
139
140 # 求一个平均值
141 p = P / test_sents_num
142 r = R / test_sents_num
143 f1 = (2 * p * r) / (p + r)
144 for i in range(test_sents_num)

```

Run: fenci

P,R,F1 0.8488645010171012 0.8938939571664417 0.8707974927325003
52.729925870895386

Process finished with exit code 0

最后使用双向切分的方法，正确率几乎不变，用时 52s，几乎翻倍的时间。

	P	R	F1	Time
前向	0.8413	0.8864	0.8864	26.91
后向	0.8482	0.8934	0.8934	27.51
双向	0.8489	0.8939	0.8708	52.73

可见，基于词典的分词方式，不论哪种方法，正确率基本稳定在这个范围上。效果还算可以。

2. 基于语料的统计分词，使用二元语法模型来构建词库，然后将句子生成词网，在使用 viterbi 算法来计算最优解，在这当中使用+1 法来处理前后之间概率。代码放在附录 2 首先我们处理原始文件，将人民日报语料库的带空格的文件处理成 csv 的表格形式方便我们进行分词的整理。

```

pythonProject - chuli.py
1 import csv
2
3 mat = []
4
5 with open('renmin.txt', 'r') as f: # 打开文件
6     for line in f:
7         mat.append([l for l in line.split()])
8
9 print(mat)
10
11 with open('renmin.csv', 'w', newline='') as csvfile:
12     writer = csv.writer(csvfile)
13     for row in mat:
14         if len(row) != 0:
15             writer.writerow(row)
16

```



```
pythonProject - tongjifenci.py
Project
pythonProject ~/Downloads
chuli.py
chulichixing.py
cixing_tongji.py
CoreNatureDictionary.csv
cws.csv
danju.csv
fenci.py
main.py
my_cws_corpus.csv
renmin.csv
renmin.txt
renmincixing.csv
renmincixing.txt
temp.csv
test.csv
tongjifenci.py
train.im
External Libraries
Scratches and Consoles
Structure
Functions

# 二元语法模型
def bgram(sents: list):
    dic = {}
    dic["#起始#"] = dict()
    for sent in sents:
        for i in range(0, len(sent) - 1):
            if sent[i] not in dic:
                dic[sent[i]] = dict()
                dic[sent[i]][sent[i + 1]] = 1
            else:
                if sent[i + 1] in dic[sent[i]]:
                    dic[sent[i]][sent[i + 1]] += 1
                else:
                    dic[sent[i]][sent[i + 1]] = 1
        if sent[0] not in dic["#起始#"]:
            dic["#起始#"][sent[0]] = 1
        else:
            dic["#起始#"][sent[0]] += 1
        if sent[len(sent) - 1] not in dic:
            dic[sent[len(sent) - 1]] = dict()
            dic[sent[len(sent) - 1]]["#结束#"] = 1
        elif "#结束#" not in dic[sent[len(sent) - 1]]:
            dic[sent[len(sent) - 1]]["#结束#"] = 1
        else:
            dic[sent[len(sent) - 1]]["#结束#"] += 1
    return dic

bi_gram = bgram(fenci_sents)

fenci_ci() calculate_weight() if word_one_in_gram
```

这里是生成句子词网的核心代码：

```
pythonProject - tongjifenci.py
Project
pythonProject ~/Downloads
chuli.py
chulichixing.py
cixing_tongji.py
CoreNatureDictionary.csv
cws.csv
danju.csv
fenci.py
main.py
my_cws_corpus.csv
renmin.csv
renmin.txt
renmincixing.csv
renmincixing.txt
temp.csv
test.csv
tongjifenci.py
train.im
External Libraries
Scratches and Consoles
Structure
Functions

def fen_ci(text):
    # 生成一元语法词网
    def generate_wordnet(gram, text):
        net = [[] for _ in range(len(text) + 2)]
        for i in range(len(text)):
            for j in range(i + 1, len(text) + 1):
                word = text[i:j]
                if word in gram:
                    net[i + 1].append(word)
        i = 1
        while i < len(net) - 1:
            if len(net[i]) == 0: # 空白行
                j = i + 1
            else:
                for j in range(i + 1, len(net) - 1):
                    # 寻找第一个非空行j
                    if len(net[j]):
                        break
                # 填补 i, j 之间的空白行
                net[i].append(text[i - 1:j] - 1)
                i = j
            else:
                i += len(net[i] - 1)
        return net
    # 测试一个句子
    si_net = generate_wordnet(si_gram, text)
    # print(si_net)

def calculate_gram_sum(gram: dict):
    num = 0
    for g in gram.keys():
        num += sum(gram[g].values())

fenci_ci() generate_wordnet() while i < len(net) - 1 if len(net[i]) == 0
```

这里是+1 平滑处理的核心代码：


```
pythonProject - tongjifenci.py
Project
pythonProject ~/Downloads
chuli.py 240
chulichixing.py 241
cixing_tongji.py 242
CoreNatureDictionary.csv 244
cws.csv 245
danju.csv 246
fenci.py 247
main.py 248
my_cws_corpus.csv 249
renmin.csv 250
renmin.txt 251
renmincixing.csv 252
renmincixing.txt 253
temp.csv 254
test.csv 255
tongjifenci.py 257
train.im
External Libraries
Scratches and Consoles

Run: tongjifenci
0.303070950373043
0.970924740212029
0.9761729820510129
0.981421223889969
0.9866694657289808
0.9919177075679647
0.9971659494069487
0.9904639853036992 0.9842781292348377 0.9873613687091717
307.9367780685425

Process finished with exit code 0
```

3. 对分词的结果进行词性标注。使用了统计的方法。此处代码同附录 2。
我们先对人民日报带词性标注的 txt 文件进行一个转换，转换成便于我操作的 csv 文件。

```
pythonProject - chulichixing.py
Project
pythonProject ~/Downloads
chuli.py 1
chulichixing.py 2
cixing_tongji.py 3
CoreNatureDictionary.csv 4
cws.csv 5
danju.csv 6
fenci.py 7
main.py 8
my_cws_corpus.csv 9
renmin.csv 10
renmin.txt 11
renmincixing.csv 12
renmincixing.txt 13
temp.csv 14
test.csv 15
tongjifenci.py 16
train.im 17
External Libraries 18
Scratches and Consoles

Run: chulichixing
import csv
mat = []
with open("renmincixing.txt", "r") as f: # 打开文件
    for line in f:
        line = line[22:]
        print(line)
        mat.append([l for l in line.split()])
print(mat)
with open("renmincixing.csv", "w", newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in mat:
        if len(row) != 0:
            writer.writerow(row)
```

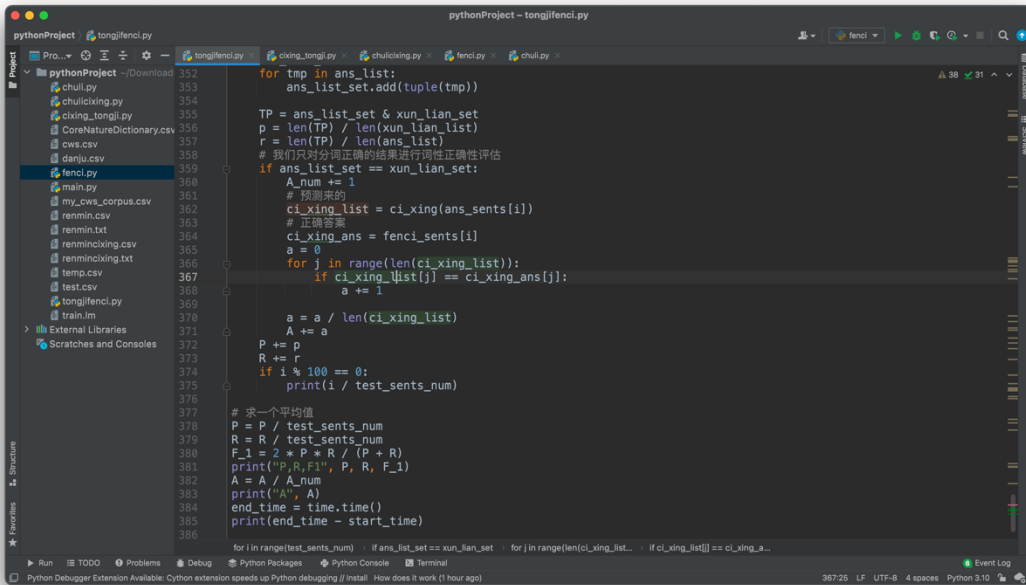


```
pythonProject - tongjifenci.py
Project
pythonProject ~/Downloads
chuli.py
chulicixing.py
cixing_tongji.py
CoreNatureDictionary.csv
cws.csv
danju.csv
fenci.py
main.py
my_cws_corpus.csv
renmin.csv
renmin.txt
renmincixing.csv
renmincixing.txt
temp.csv
test.csv
tongjifenci.py
train.im
External Libraries
Scratches and Consoles
Structure
100 part = dict()
101 # 统计所有词性个数
102 for sent in cixing_sents:
103     for word in sent:
104         word_part = word.split('/')[-1].split('')[0].split('')[0]
105         if word_part in part:
106             part[word_part] += 1
107         else:
108             part[word_part] = 1
109
110 part_len = len(part)
111 print(part, "一共多少类:", part_len)
112 print("总频次", sum(part.values()))
113
114 # 取得转移矩阵
115 trans = dict()
116 for sent in cixing_sents:
117     for i in range(len(sent) - 1):
118         one = sent[i].split('/')[-1].split('')[0].split('')[0]
119         two = sent[i + 1].split('/')[-1].split('')[0].split('')[0]
120         if one in trans:
121             if two in trans[one]:
122                 trans[one][two] += 1
123             else:
124                 trans[one][two] = 1
125         else:
126             trans[one] = dict()
127             trans[one][two] = 1
128 print(trans)
129
130 # 每个词的词性概率
131 percent = dict()
132 for sent in cixing_sents:
133     for word in sent:
134         word_part = word.split('/')[-1].split('')[0].split('')[0]
135         fen_ci()
136         calculate_weight()
137         if word_one in gram:
138             else:
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
```

这里是推测词性的核心代码：

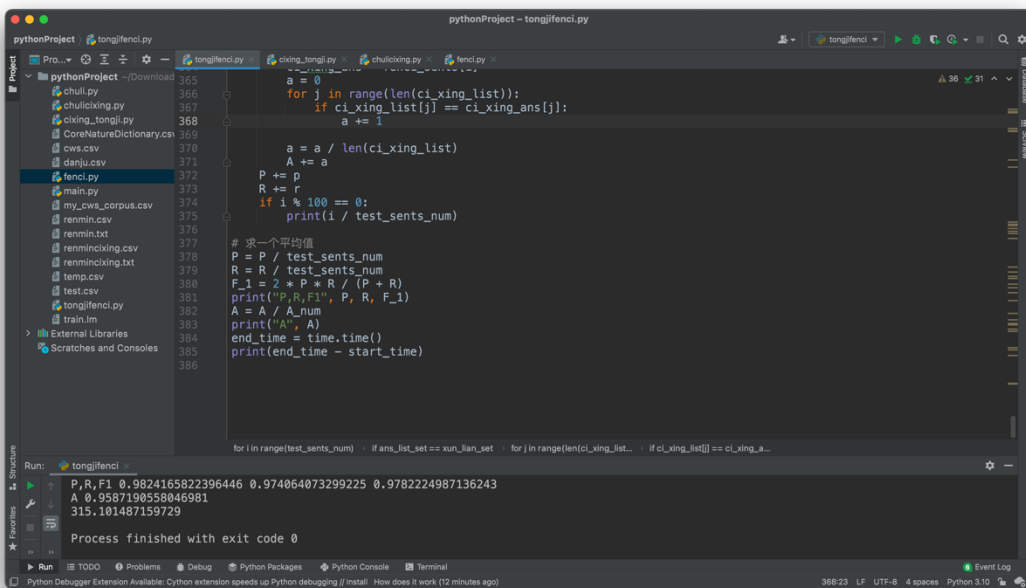
```
pythonProject - tongjifenci.py
Project
pythonProject ~/Downloads
chuli.py
chulicixing.py
cixing_tongji.py
CoreNatureDictionary.csv
cws.csv
danju.csv
fenci.py
main.py
my_cws_corpus.csv
renmin.csv
renmin.txt
renmincixing.csv
renmincixing.txt
temp.csv
test.csv
tongjifenci.py
train.im
External Libraries
Scratches and Consoles
Structure
247 def ci_xing(text):
248     text_percent = []
249
250     # 这里我们假设是所有单词都已经人民日报语料库了
251     for word in text:
252         word_percent = percent[word]
253         text_percent.append(word_percent)
254
255     # print(text_percent)
256
257     # 下面我们来使用Viterbi算法计算出最佳的组成
258     dis = [dict() for _ in range(len(text))]
259     node = [dict() for _ in range(len(text))]
260     for first in text_percent[0].keys():
261         dis[0][first] = 1
262     for i in range(len(text) - 1):
263         word_one = text[i]
264         word_two = text[i + 1]
265         word_one_percent_dict = text_percent[i]
266         word_two_percent_dict = text_percent[i + 1]
267
268         word_one_percent_key = list(word_one_percent_dict.keys())
269         word_one_percent_value = list(word_one_percent_dict.values())
270         word_two_percent_key = list(word_two_percent_dict.keys())
271         word_two_percent_value = list(word_two_percent_dict.values())
272         for word_two_per in word_two_percent_key:
273             tmp_dis = []
274             for word_one_per in word_one_percent_key:
275                 if word_two_per in trans[word_one_per]:
276                     tmp_num = dis[i][word_one_per] * (
277                         (trans[word_one_per][word_two_per] + 1) / (part[word_one_per] + part_len)) * (
278                             text_percent[i + 1][word_two_per] / part[word_two_per])
279                     tmp_dis.append(tmp_num)
280             else:
281                 tmp_dis.append(0)
282         dis[i + 1][word_two_per] = max(tmp_dis)
283         node[i + 1][word_two_per] = word_one_per
284
285     for i in range(len(text) - 1):
286         for word_two_per in word_two_per_key:
287             for word_one_per in word_one_per_key:
288                 if word_two_per in trans[word_one_per]:
289                     dis[i + 1][word_two_per] = dis[i][word_one_per] * (
290                         (trans[word_one_per][word_two_per] + 1) / (part[word_one_per] + part_len)) * (
291                             text_percent[i + 1][word_two_per] / part[word_two_per])
292                     node[i + 1][word_two_per] = word_one_per
293
294     return node
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
```

下面是对分词结果的正确性评估，这里我使用了书中使用的方法，仅计算一个 Accuracy 作为正确率评估标准，同时我只对之前分词正确的结果结果进行词性评估，这样可以避免其它的错误。



```
pythonProject - tongjifenci.py
Project
  pythonProject ~/Downloads
  chuli.py
  chuliciking.py
  ciking_tongji.py
  CoreNatureDictionary.csv
  cws.csv
  danju.csv
  fenci.py
  main.py
  my_cws_corpus.csv
  renmin.csv
  renmin.txt
  renminciking.csv
  renminciking.txt
  temp.csv
  test.csv
  tongjifenci.py
  train.im
  External Libraries
  Scratches and Consoles
  Structure
  Favorites
  pythonProject - tongjifenci.py
  fenci.py
  TP = ans_list_set & xun_lian_set
  p = len(TP) / len(xun_lian_list)
  r = len(TP) / len(ans_list)
  # 我们对分词正确的结果进行词性正确性评估
  if ans_list_set == xun_lian_set:
      A_num += 1
      # 预测来的
      ci_xing_list = ci_xing(ans_sents[i])
      # 正确答案
      ci_xing_ans = fenci_sents[i]
      a = 0
      for j in range(len(ci_xing_list)):
          if ci_xing_list[j] == ci_xing_ans[j]:
              a += 1
      a = a / len(ci_xing_list)
      A += a
  P += p
  R += r
  if i % 100 == 0:
      print(i / test_sents_num)
  # 求一个平均值
  P = P / test_sents_num
  R = R / test_sents_num
  F_1 = 2 * P * R / (P + R)
  print("P,R,F1", P, R, F_1)
  A = A / A_num
  print("A", A)
  end_time = time.time()
  print(end_time - start_time)
  Run
  Run: P,R,F1 0.9587159558946981 0.974064073299225 0.9782224987136243
  A 0.9587159558946981
  315.101487159729
  Process finished with exit code 0
```

通过检测结果来看，词性标注的正确率大概在 95%左右，同时共计耗时 315s。还算一个相对不错的效果。

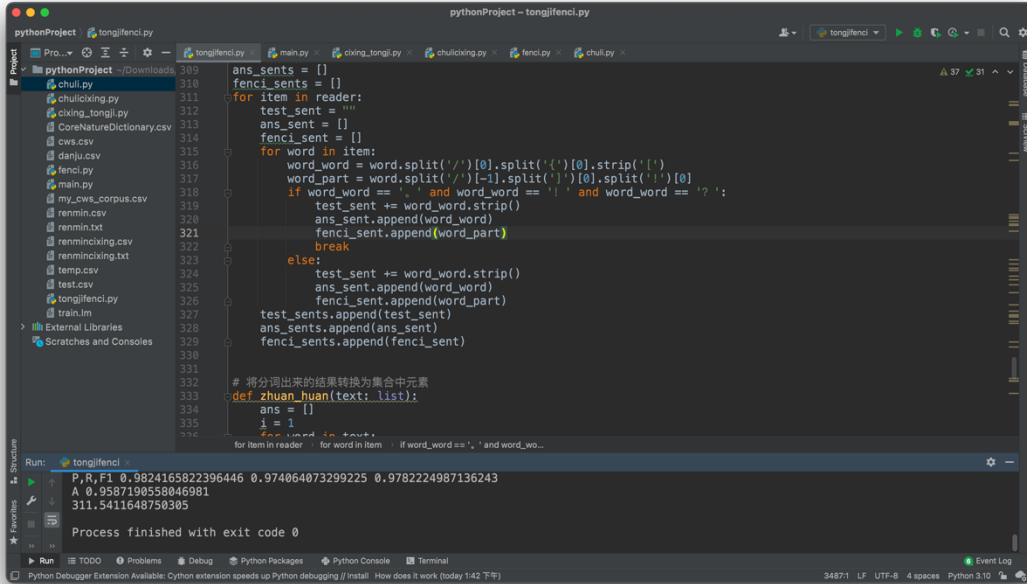


```
pythonProject - tongjifenci.py
Project
  pythonProject ~/Downloads
  chuli.py
  chuliciking.py
  ciking_tongji.py
  CoreNatureDictionary.csv
  cws.csv
  danju.csv
  fenci.py
  main.py
  my_cws_corpus.csv
  renmin.csv
  renmin.txt
  renminciking.csv
  renminciking.txt
  temp.csv
  test.csv
  tongjifenci.py
  train.im
  External Libraries
  Scratches and Consoles
  Structure
  Favorites
  pythonProject - tongjifenci.py
  fenci.py
  a = 0
  for j in range(len(ci_xing_list)):
      if ci_xing_list[j] == ci_xing_ans[j]:
          a += 1
  a = a / len(ci_xing_list)
  A += a
  P += p
  R += r
  if i % 100 == 0:
      print(i / test_sents_num)
  # 求一个平均值
  P = P / test_sents_num
  R = R / test_sents_num
  F_1 = 2 * P * R / (P + R)
  print("P,R,F1", P, R, F_1)
  A = A / A_num
  print("A", A)
  end_time = time.time()
  print(end_time - start_time)
  Run
  Run: P,R,F1 0.9587159558946981 0.974064073299225 0.9782224987136243
  A 0.9587159558946981
  315.101487159729
  Process finished with exit code 0
```

四、实验遇到的问题与麻烦

1. 首先就遇到的难题是处理人民日报那个 txt，一开始没有想到很方便的办法把它处理为 list 数据，最后通过转换为 csv 格式，在直接导入到 list 当中确实简化了不少步骤。
2. 处理二元语法模型的时候，这个“#始始#”，“#末末#”的处理确实费了不少功夫，总是在这里缺少一些项目，debug 了很长时间。
3. Viterbi 算法中存在的字典中缺少的值如何去补充，不存在的转化概率该如何去补充。

4. 这个也是尚未处理的问题，我做预测的时候仍然采用原人民日报的换行格式，即一行就为一个句子，但这个很明显并不是我们日常意义上所理解的句子，我们通常理解的句子中，是以“。”，“！”，“？”作为句子结尾的。我重新刷洗了数据，并让句子按照这三个标点符号作为结尾来预测数据。



最终得出的训练结果中，正确率不变，说明我们能很好的将标点符号分开，但是时间却缩小了一些，说明缩短句子确实有助于减少大规模词网的运算量。

五、附录

附录一：基于词典的分词方法：

```
import csv
import time
```

```
start_time = time.time()
```

```
# 读入字典
```

```
def load_dictionary():
    word_list = set()
    csvFile = open("test.csv", "r")
    reader = csv.reader(csvFile)
    for item in reader:
        word_list.add(item[0])
    return word_list
```

```
# 完全切分式中文分词
```

```
# 如果在词典中则认为是一个词
```

```
def fully_segment(text, dic):
```

```

word_list = []
for i in range(len(text)):
    for j in range(i + 1, len(text) + 1):
        word = text[i:j]
        if word in dic:
            word_list.append(word)
return word_list

```

正向最长匹配

从当前扫描位置的单字所有可能的结尾，我们找最长的

```

def forward_segment(text, dic):
    word_list = []
    i = 0
    while i < len(text):
        longest_word = text[i]
        for j in range(i + 1, len(text) + 1):
            word = text[i:j]
            if word in dic:
                if len(word) > len(longest_word):
                    longest_word = word
        word_list.append(longest_word)
        i += len(longest_word)
    return word_list

```

逆向最长匹配

```

def back_segment(text, dic):
    word_list = []
    i = len(text) - 1
    while i >= 0:
        longest_word = text[i]
        for j in range(0, i):
            word = text[j:i + 1]
            if word in dic:
                if len(word) > len(longest_word):
                    longest_word = word
        word_list.append(longest_word)
        i -= len(longest_word)
    word_list.reverse()
    return word_list

```

统计单字成词的个数

```

def count_single_char(word_list: list):
    return sum(1 for word in word_list if len(word) == 1)

# 双向最长匹配，实际上是做了个融合
def bidirectional_segment(text, dic):
    f = forward_segment(text, dic)
    b = back_segment(text, dic)
    # 词数更少更优先
    if len(f) < len(b):
        return f
    elif len(f) > len(b):
        return b
    else:
        # 单字更少更优先
        if count_single_char(f) < count_single_char(b):
            return f
        else: # 都相等时我们更倾向于逆向匹配的
            return b

dic = load_dictionary()
print(len(dic))
print("完全切分：", fully_segment("就读于北京大学", dic))
print("前向切分：", forward_segment("就读于北京大学", dic))
print("前向切分：", forward_segment("研究生物起源", dic))
print("后向切分：", back_segment("研究生物起源", dic))
print("双向切分：", bidirectional_segment("研究生物起源", dic))

test_file = open("renmin.csv", "r")
reader = csv.reader(test_file)
test_sents = []
ans_sents = []
for item in reader:
    test_sent = ""
    ans_sent = []
    for word in item:
        test_sent += word.strip()
        ans_sent.append(word)
    test_sents.append(test_sent)
    ans_sents.append(ans_sent)

# 将分词出来的结果转换为集合中元素

```

```

def zhuan_huan(text: list):
    ans = []
    i = 1
    for word in text:
        ans.append([i, i + len(word) - 1])
        i += len(word)
    return ans

test_sents_num = len(test_sents)
print(test_sents_num)
P = 0
R = 0
for i in range(test_sents_num):
    xun_lian = bidirectional_segment(test_sents[i], dic)
    xun_lian_list = zhuan_huan(xun_lian)
    ans_list = zhuan_huan(ans_sents[i])

    xun_lian_set = set()
    for tmp in xun_lian_list:
        xun_lian_set.add(tuple(tmp))
    ans_list_set = set()
    for tmp in ans_list:
        ans_list_set.add(tuple(tmp))
    TP = ans_list_set & xun_lian_set
    p = len(TP) / len(xun_lian_list)
    r = len(TP) / len(ans_list)
    P += p
    R += r
    if i % 100 == 0:
        print(i / test_sents_num)

# 求一个平均值
P = P / test_sents_num
R = R / test_sents_num
F_1 = 2 * P * R / (P + R)
print("P,R,F1", P, R, F_1)
end_time = time.time()
print(end_time - start_time)

```

附录二：基于统计的分词和词性标注

```

import csv
import time

```

```

start_time = time.time()
fenci_file = open("renmin.csv", "r")
fenci_reader = csv.reader(fenci_file)
fenci_sents = []
for item in fenci_reader:
    sent = []
    for i in item:
        sent.append(i.strip())
    fenci_sents.append(sent)

sum_zero = 0
for sent in fenci_sents:
    if len(sent) == 0:
        sum_zero += 1
    print(sent)
print(sum_zero)

```

一元语法模型

```

def sgram(sents: list):
    dic = {}
    dic['#起始#'] = 0
    dic['#末末#'] = 0
    for sent in sents:
        for item in sent:
            if item in dic:
                dic[item] = dic[item] + 1
            else:
                dic[item] = 1
        dic['#起始#'] += 1
        dic['#末末#'] += 1
    return dic

```

```

si_gram = sgram(fenci_sents)

```

```

# print(si_gram)

```

二元语法模型

```

def bgram(sents: list):
    dic = {}
    dic["#起始#"] = dict()

```



```

for sent in sents:
    for i in range(0, len(sent) - 1):
        if sent[i] not in dic:
            dic[sent[i]] = dict()
            dic[sent[i]][sent[i + 1]] = 1
        else:
            if sent[i + 1] in dic[sent[i]]:
                dic[sent[i]][sent[i + 1]] += 1
            else:
                dic[sent[i]][sent[i + 1]] = 1

    if sent[0] not in dic["#起始#"]:
        dic["#起始#"][sent[0]] = 1
    else:
        dic["#起始#"][sent[0]] += 1

    if sent[len(sent) - 1] not in dic:
        dic[sent[len(sent) - 1]] = dict()
        dic[sent[len(sent) - 1]]["#末末#"] = 1
    elif "#末末#" not in dic[sent[len(sent) - 1]]:
        dic[sent[len(sent) - 1]]["#末末#"] = 1
    else:
        dic[sent[len(sent) - 1]]["#末末#"] += 1
return dic

```

```
bi_gram = bgram(fenci_sents)
```

```
# print(bi_gram)
```

```
# 查看一元词频
```

```
def select_si_gram(gram: dict, word):
    return gram[word]
```

```
def selecr_bi_gram(gram: dict, word_one, word_two):
    return gram[word_one][word_two]
```

```
cixing_file = open("renmincixing.csv", "r")
cixing_reader = csv.reader(cixing_file)
cixing_sents = []

```

```

for item in cixing_reader:
    sent = []
    for i in item:
        sent.append(i.strip())
    cixing_sents.append(sent)

# print(sents)

part = dict()
# 统计所有词性个数
for sent in cixing_sents:
    for word in sent:
        word_part = word.split('/')[-1].split('')[0].split('!')[0]
        if word_part in part:
            part[word_part] += 1
        else:
            part[word_part] = 1

part_len = len(part)
print(part, "一共多少类:", part_len)
print("总频次", sum(part.values()))

# 或得转移矩阵
trans = dict()
for sent in cixing_sents:
    for i in range(len(sent) - 1):
        one = sent[i].split('/')[-1].split('')[0].split('!')[0]
        two = sent[i + 1].split('/')[-1].split('')[0].split('!')[0]
        if one in trans:
            if two in trans[one]:
                trans[one][two] += 1
            else:
                trans[one][two] = 1
        else:
            trans[one] = dict()
            trans[one][two] = 1
print(trans)

# 每个词的词性概率
percent = dict()
for sent in cixing_sents:
    for word in sent:
        word_word = word.split('/')[0].split('{')[0].strip('[]')
        word_part = word.split('/')[-1].split('')[0].split('!')[0]

```

```

    if word_word in percent:
        if word_part in percent[word_word]:
            percent[word_word][word_part] += 1
        else:
            percent[word_word][word_part] = 1
    else:
        percent[word_word] = dict()
        percent[word_word][word_part] = 1

# print(percent)

def fen_ci(text):
    # 生成一元语法词网
    def generate_wordnet(gram, text):
        net = [[] for _ in range(len(text) + 2)]
        for i in range(len(text)):
            for j in range(i + 1, len(text) + 1):
                word = text[i:j]
                if word in gram:
                    net[i + 1].append(word)
        i = 1
        while i < len(net) - 1:
            if len(net[i]) == 0: # 空白行
                j = i + 1

                for j in range(i + 1, len(net) - 1):
                    # 寻找第一个非空行j
                    if len(net[j]):
                        break
                # 填补i, j之间的空白行
                net[i].append(text[i - 1:j - 1])
                i = j
            else:
                i += len(net[i][-1])
        return net

# 测试一个句子
si_net = generate_wordnet(si_gram, text)

# print(si_net)

def calculate_gram_sum(gram: dict):
    num = 0

```

```

    for g in gram.keys():
        num += sum(gram[g].values())
    return num

# 计算word_one后出现word_two的概率, 带上+1平滑处理
def calculate_weight(gram: dict, word_one, word_two, gram_sum):
    if word_one in gram:
        word_one_all = gram[word_one].values()
        if word_two in gram[word_one]:
            return (gram[word_one][word_two] + 1) / (sum(word_one_all) +
gram_sum)
        else:
            return 1 / (sum(word_one_all) + gram_sum)
    else:
        return 1 / gram_sum

bi_gram_sum = calculate_gram_sum(bi_gram)

# print(bi_gram_sum)

def viterbi(wordnet):
    dis = [dict() for _ in range(len(wordnet))]
    node = [dict() for _ in range(len(wordnet))]
    word_line = [dict() for _ in range(len(wordnet))]
    wordnet[len(wordnet) - 1].append("#末末#")
    # 更新第一行
    for word in wordnet[1]:
        dis[1][word] = calculate_weight(bi_gram, "#起始#", word,
bi_gram_sum)
        node[1][word] = 0
        word_line[1][word] = "#起始#"
    # 遍历每一行wordnet
    for i in range(1, len(wordnet) - 1):
        # 遍历每一行中单词
        for word in wordnet[i]:
            # 更新加上这个单词的距离之后那个位置的所有单词的距离
            for to in wordnet[i + len(word)]:
                if word in dis[i]:
                    if to in dis[i + len(word)]:
                        # 要的是最大的概率
                        if dis[i + len(word)][to] < dis[i][word] *
calculate_weight(bi_gram, word, to, bi_gram_sum):
                            dis[i + len(word)][to] = dis[i][word] *
calculate_weight(bi_gram, word, to, bi_gram_sum)

```

```

        node[i + len(word)][to] = i
        word_line[i + len(word)][to] = word
    else:
        dis[i + len(word)][to] = dis[i][word] *
calculate_weight(bi_gram, word, to, bi_gram_sum)
        node[i + len(word)][to] = i
        word_line[i + len(word)][to] = word

# 回溯
path = []
f = node[len(node) - 1]["#末末#"]
fword = word_line[len(word_line) - 1]["#末末#"]
path.append(fword)
while f:
    tmpword = fword
    fword = word_line[f][tmpword]
    f = node[f][tmpword]
    path.append(fword)
path = path[:-1]
path.reverse()
return dis, node, word_line, path

(dis, _, _, path) = viterbi(si_net)
# print(dis)
# print(path)
return path

```

```

def ci_xing(text):
    text_percent = []

    # 这里我们假设是所有单词都已经人民日报语料库了
    for word in text:
        word_percent = percent[word]
        text_percent.append(word_percent)

    # print(text_percent)

    # 下面我们来使用Viterbi算法计算出最佳的组成
    dis = [dict() for _ in range(len(text))]
    node = [dict() for _ in range(len(text))]
    for first in text_percent[0].keys():
        dis[0][first] = 1
    for i in range(len(text) - 1):

```

```

word_one = text[i]
word_two = text[i + 1]
word_one_percent_dict = text_percent[i]
word_two_percent_dict = text_percent[i + 1]

word_one_percent_key = list(word_one_percent_dict.keys())
word_one_percent_value = list(word_one_percent_dict.values())
word_two_percent_key = list(word_two_percent_dict.keys())
word_two_percent_value = list(word_two_percent_dict.values())
for word_two_per in word_two_percent_key:
    tmp_dis = []
    for word_one_per in word_one_percent_key:
        if word_two_per in trans[word_one_per]:
            tmp_num = dis[i][word_one_per] * (
                (trans[word_one_per][word_two_per] + 1) /
                (part[word_one_per] + part_len)) * (
                    text_percent[i + 1][word_two_per] /
                    part[word_two_per])
            tmp_dis.append(tmp_num)
        else:
            tmp_num = dis[i][word_one_per] * (1 / (part[word_one_per] +
                part_len)) * (
                    text_percent[i + 1][word_two_per] /
                    part[word_two_per])
            tmp_dis.append(tmp_num)

    max_tmp_dis = max(tmp_dis)
    max_tmp_dis_loc = tmp_dis.index(max_tmp_dis)
    dis[i + 1][word_two_per] = max_tmp_dis
    node[i + 1][word_two_per] = word_one_percent_key[max_tmp_dis_loc]
# print(dis, node)

# 根据node来倒找答案
path = []
f_value = list(dis[len(dis) - 1].values())
f_key = list(dis[len(dis) - 1].keys())
f = f_key[f_value.index(max(f_value))]

path.append(f)
for i in range(len(dis) - 1, 0, -1):
    f = node[i][f]
    path.append(f)
path.reverse()
return path

```

```

# 对所有训练集进行测试
test_file = open("renmincixing.csv", "r")
reader = csv.reader(test_file)
test_sents = []
ans_sents = []
fenci_sents = []
for item in reader:
    test_sent = ""
    ans_sent = []
    fenci_sent = []
    for word in item:
        word_word = word.split('/')[0].split('{')[0].strip('{')
        word_part = word.split('/')[-1].split('}')[0].split('!')[0]
        if word_word == '。' and word_word == '!' and word_word == '?':
            test_sent += word_word.strip()
            ans_sent.append(word_word)
            fenci_sent.append(word_part)
            break
        else:
            test_sent += word_word.strip()
            ans_sent.append(word_word)
            fenci_sent.append(word_part)
    test_sents.append(test_sent)
    ans_sents.append(ans_sent)
    fenci_sents.append(fenci_sent)

```

将分词出来的结果转换为集合中元素

```

def zhuan_huan(text: list):
    ans = []
    i = 1
    for word in text:
        ans.append([i, i + len(word) - 1])
        i += len(word)
    return ans

```

```

test_sents_num = len(test_sents)
print(test_sents_num)
P = 0
R = 0
A = 0

```

```

A_num = 0
for i in range(test_sents_num):
    xun_lian = fen_ci(test_sents[i])
    xun_lian_list = zhuan_huan(xun_lian)
    ans_list = zhuan_huan(ans_sents[i])

    xun_lian_set = set()
    for tmp in xun_lian_list:
        xun_lian_set.add(tuple(tmp))

    ans_list_set = set()
    for tmp in ans_list:
        ans_list_set.add(tuple(tmp))

    TP = ans_list_set & xun_lian_set
    p = len(TP) / len(xun_lian_list)
    r = len(TP) / len(ans_list)
    # 我们只对分词正确的结果进行词性正确性评估
    if ans_list_set == xun_lian_set:
        A_num += 1
        # 预测来的
        ci_xing_list = ci_xing(ans_sents[i])
        # 正确答案
        ci_xing_ans = fenci_sents[i]
        a = 0
        for j in range(len(ci_xing_list)):
            if ci_xing_list[j] == ci_xing_ans[j]:
                a += 1

        a = a / len(ci_xing_list)
        A += a
    P += p
    R += r
    if i % 100 == 0:
        print(i / test_sents_num)

# 求一个平均值
P = P / test_sents_num
R = R / test_sents_num
F_1 = 2 * P * R / (P + R)
print("P,R,F1", P, R, F_1)
A = A / A_num
print("A", A)
end_time = time.time()

```



```
print(end_time - start_time)
```

附录三：处理人民日报分词数据：

```
import csv

mat = []

with open("renmin.txt", "r") as f: # 打开文件
    for line in f:
        mat.append([l for l in line.split()])

print(mat)

with open('renmin.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in mat:
        if len(row) != 0:
            writer.writerow(row)
```

附录四：处理人民日报词性数据：

```
import csv

mat = []

with open("renmincixing.txt", "r") as f: # 打开文件
    for line in f:
        line = line[22:]
        print(line)
        mat.append([l for l in line.split()])

print(mat)

with open('renmincixing.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    for row in mat:
        if len(row) != 0:
            writer.writerow(row)
```