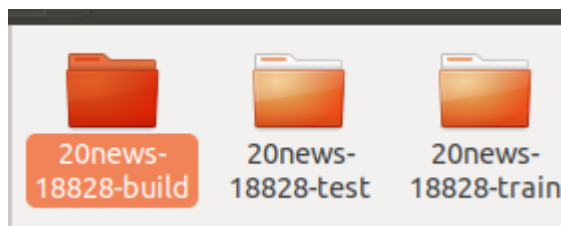


实验步骤：将数据集解压后，将其中的 80%作为训练集，20%作为测试集。



预处理时报错，缺少所需的 stopwords:

```
liubuntu@liubuntu:~/Desktop$ python knn.py
Traceback (most recent call last):
  File "/usr/local/lib/python3.6/dist-packages/nltk/corpus/util.py", line 80, in __load
    try: root = nltk.data.find('{}{}'.format(self.subdir, zip_name))
  File "/usr/local/lib/python3.6/dist-packages/nltk/data.py", line 675, in find
    raise LookupError(resource_not_found)
LookupError:
*****
Resource stopwords not found.
Please use the NLTK Downloader to obtain the resource:

>>> import nltk
>>> nltk.download('stopwords')

Searched in:
- '/home/liubuntu/nltk_data'
- '/usr/share/nltk_data'
- '/usr/local/share/nltk_data'
- '/usr/lib/nltk_data'
- '/usr/local/lib/nltk_data'
- '/usr/nltk_data'
- '/usr/share/nltk_data'
- '/usr/lib/nltk_data'
*****
```

解决办法:

```
liubuntu@liubuntu:~/Desktop$ python knn.py
[nltk_data] Downloading package stopwords to
[nltk_data] /home/liubuntu/nltk_data...
[nltk_data] Unzipping corpora/stopwords.zip.
Traceback (most recent call last):
```

python 读取文件时发生了错误

```
File "knn.py", line 207, in <module>
    main()
File "knn.py", line 199, in main
    preProcess()
File "knn.py", line 25, in preProcess
    preProcessFile(dirs[i],files[j])
File "knn.py", line 34, in preProcessFile
    data = open(src,"r").readlines()
File "/usr/lib/python3.6/codecs.py", line 321, in decode
    (result, consumed) = self._buffer_decode(data, self.errors, final)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xfd in position 293: invalid start byte
```

解决办法:

将

```
data = open(src,"r").readlines()
```

改为:

```
data = open(src,"rb").readlines()
```

单个文件的 VSM 结果:

```

18:athelstan
testvector /home/lubuntu/20news-18828-test/atl.athelstan/5119 (subject: 1.5, 'gospel': 3.0, 'edu': 3.0, 'write': 1.5, 'well': 1.5, 'luke': 12.0, 'new': 1.5, 'matthew': 3.0, 'sourc': 3.0, 'assum': 1.5, 'put': 7
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
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:
110:
111:
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:

```

运行结果：

```
liubuntu@liubuntu:~/Desktop$ python knn.py
[nltk_data] Downloading package stopwords to
[nltk_data] /home/liubuntu/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
correctness is 0.7659574468085106
```

代码原理:

### 1、预处理:

#创建新文件夹，存放预处理后的文本数据

```
def preProcess():
```

```
dirs=os.listdir(rootDir)
```

```
for i in range(len(dirs)):
```

```
if i == 0:
```

continue

```
fileDir=rootDir+dirs[i]
```

```
buildFileDir=buildDir+dires[i]
```

```
files=os.listdir(fileDir)
```

```
if os.path.exists(buildFileDir)==False:
```

```
os.mkdir(buildFileDir)
```

```
for j in range(len(files)):
```

```
preProcessFile(dirs[i],files[j])
```

### #建立目标文件夹，生成目标文件

```
def preProcessFile(dir,fileName):
```

```
src = rootDir+ "/" + dir + "/" + fileName
```

```
dst = buildDir + "/" + dir + "/" + fileName
```

```
dstFile = open(dst,'w')
```

```
data = open(src,"rb").readlines()
```

```
for line in data:
```

```
result = processLine(line)
```

for word in result:

```
dstFile.write('%s/n' % word)
```

```
dstFile.close()
```

```
#对每行字符串进行处理，主要是去除非字母字符，转换大写为小写，去除停用词
def processLine(line):
    stopwords = nltk.corpus.stopwords.words('english') #去停用词
    porter = nltk.PorterStemmer() #词干分析
    splitter = re.compile('[^a-zA-Z]') #去除非字母字符，形成分隔
    for i in range(len(line)):
        if chr(line[i]).isalpha():
            string+=chr(line[i])
        else:
            string+=" "

    words = [porter.stem(word.lower()) for word in splitter.split(string)/
              if len(word)>0 and/
              word.lower() not in stopwords]
    return words
```

## 2、建立 vsm

# 统计每个词的总的出现次数

```
def countWords():
    wordMap = {}
    wordMap2 = {}

    dirs = os.listdir(buildDir)
    for i in range(len(dirs)):
        filedir = buildDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            for line in open(filePath).readlines():
                word=line.strip('/n')
                wordMap[word]=wordMap.get(word,0) + 1
    for key, value in wordMap.items():
        if value > 10:#去掉出现次数小于 10 的词
            wordMap2[key]=value

    return wordMap2
```

#计算 idf

```

def IDF():
    wordFileMap = {}
    IDFMap = {}
    dirs = os.listdir(buildDir)
    for i in range(len(dirs)):
        filedir = buildDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            for line in open(filePath).readlines():
                word=line.strip('/n')

                if wordFileMap.get(word)== None:#该词在所有文件里第一次出现
                    wordFileMap[word]=[]
                    wordFileMap[word].append(filePath)
                elif filePath in wordFileMap[word] == False: #该词在该文件里第一次出现
                    wordFileMap[word].append(filePath)

    for word in wordFileMap:
        IDFMap[word] = 20000.0/len(wordFileMap[word]) #文件总数/该词出现的文件个数
    return IDFMap

#计算一个文件的向量
def Vector(file,wordMap,IDFMap):
    fileVector = {}
    for line in open(file).readlines():
        word=line.strip('/n')
        if word not in wordMap:
            continue
        if word not in fileVector:
            fileVector[word]=1
        else:
            fileVector[word]+=1

    for key in fileVector:
        fileVector[key]=fileVector[key]*IDFMap[word]
    return fileVector

```

### 3、KNN 分类器

#建立训练集的 map{向量： 分类}

```

def vectorMap(wordMap,IDFMap):
    vectorMap={}
    dirs = os.listdir(buildDir)
    for i in range(len(dirs)):
        filedir = buildDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            fileVector = Vector(filePath,wordMap,IDFMap)
            vectorMap[fileVector]=filedir

    return vectorMap

```

#计算两个向量的相似度

```

def similarity(trainVector,testVector):
    trainlist = []
    testlist = []
    for word in testVector:
        if word in trainVector:
            trainlist.append(trainVector[word])
            testlist.append(testlist[word])

    trainVect = numpy.mat(trainlist)
    testVect = numpy.mat(testlist)

    num = float(testVect * trainVect.T)
    denom = numpy.linalg.norm(testVect) * numpy.linalg.norm(trainVect)
    return float(num)/(1.0+float(denom))

```

#根据模型预测测试文件的分类

```

def testFileSimilarity(testFile,wordMap,IDFMap,trainVectors):
    similarityMap = {}
    testVectoer = Vector(testFile,wordMap,IDFMap)
    resultMap = {}
    for vector in trainVectors:
        sim = similarity(vector,testVectoer)
        similarityMap[sim] = trainVectors[vector]

    keys = similarityMap.keys()
    keys.sort()
    sortedResult = [(k,similarityMap[k]) for k in sorted(similarityMap.keys())]
    count = 0

    for i in sortedResult:

```

```

    if i[1] not in resultMap:
        resultMap[i[1]] = 1
    else:
        resultMap[i[1]] += 1
    count = count + 1
    if count > 100:#选择的 K 值为 100
        break

max = 0
doctype = ""
for key,value in resultMap:
    if value>max:
        doctype = key
        max = value

return doctype

```

#### 4、观察在测试集上运行的准确率

#预测文件的分类并计算正确度

```

def predictTestFileCorr(wordMap,IDFMap,trainVectors):
    dirs = os.listdir(testDir)

    for i in range(len(dirs)):
        filedir = testDir + "/" + dirs[i]
        files = os.listdir(filedir)
        for j in range(len(files)):
            filePath = filedir + "/" + files[j]
            fileNumber = fileNumber + 1
            predict = testFileSimilarity(filePath,wordMap,IDFMap,trainVectors)
            if predict == dirs[i]:
                correctCount = correctCount + 1

    return float(correctCount/fileNumber)

def main():
    preProcess()
    wordMap = countWords()
    IDFMap= IDF()
    vectors = vectorMap(wordMap,IDFMap)
    correctness = predictTestFileCorr(wordMap,IDFMap,vectors)
    print(correctness)

```