



NLP - TP1 REPORT

Prepared by EL HARRARI ANAS
Mai, 08 2023

This report consists of a detailed explanation of the written code and an overview of the manipulation objectives

Table of Contents

Page 3

Introduction

Page 4

Exercice1 : the Levenshtein Distance

Page 6

Exercice2 : NLTK and Spacy

Page 13

Comparison

Page 16

Conclusion

Introduction

°Natural language processing is an interdisciplinary subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, particularly how to program computers to process and analyze large amounts of natural language data.

In Natural language processing, the natural language is considered an essential part in order to generate an efficient algorithm. The process of treating the natural language consists of several steps that take into consideration the language itself, the type of speech and the corpus

One of the first fundamental steps in NLP processing is the pre-treatment of the text; several processes include (cleaning, stemming, tagging and so on). This manipulation aims, as a principal objectification to get familiar with this kind of treatment

In this first manipulation, we will go through some of the most important things in the process of the pre-treatment of the corpus (the aimed text), this manipulation contains two essential parts, the first one is an overview of the Levenshtein distance as it is considered as a great tool in NLP processing. For the second part, we will be going through two of the most interesting and fundamental frameworks in NLP processing in general and the treatment of text and strings in particular: NLTK and spacy which will be described in detail later in this report

this report contains the details considering the manipulations presented in the attached code and a description of the major purposes of this lab.

° <https://en.wikipedia.org/>

Exercice1 : The Levenshtein distance

Overview

In information theory, linguistics, and computer science, the Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits required to change one word into the other.

In example words, the Levenshtein distance is a way to measure how two different strings are, which means the minimum edits that are required so as to change a word to another. When normalized, the Levenshtein distance can be used to easily validate a match between listed proper names in a machine learning result.

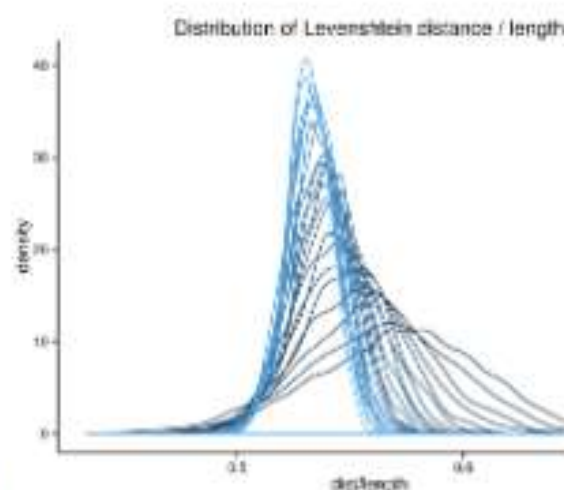
The Levenshtein distance is a theory in algebra that gives a great application in linguistic processing and NLP.

Definition [edit]

Mathematically, the Levenshtein distance between two strings a , b is given by:

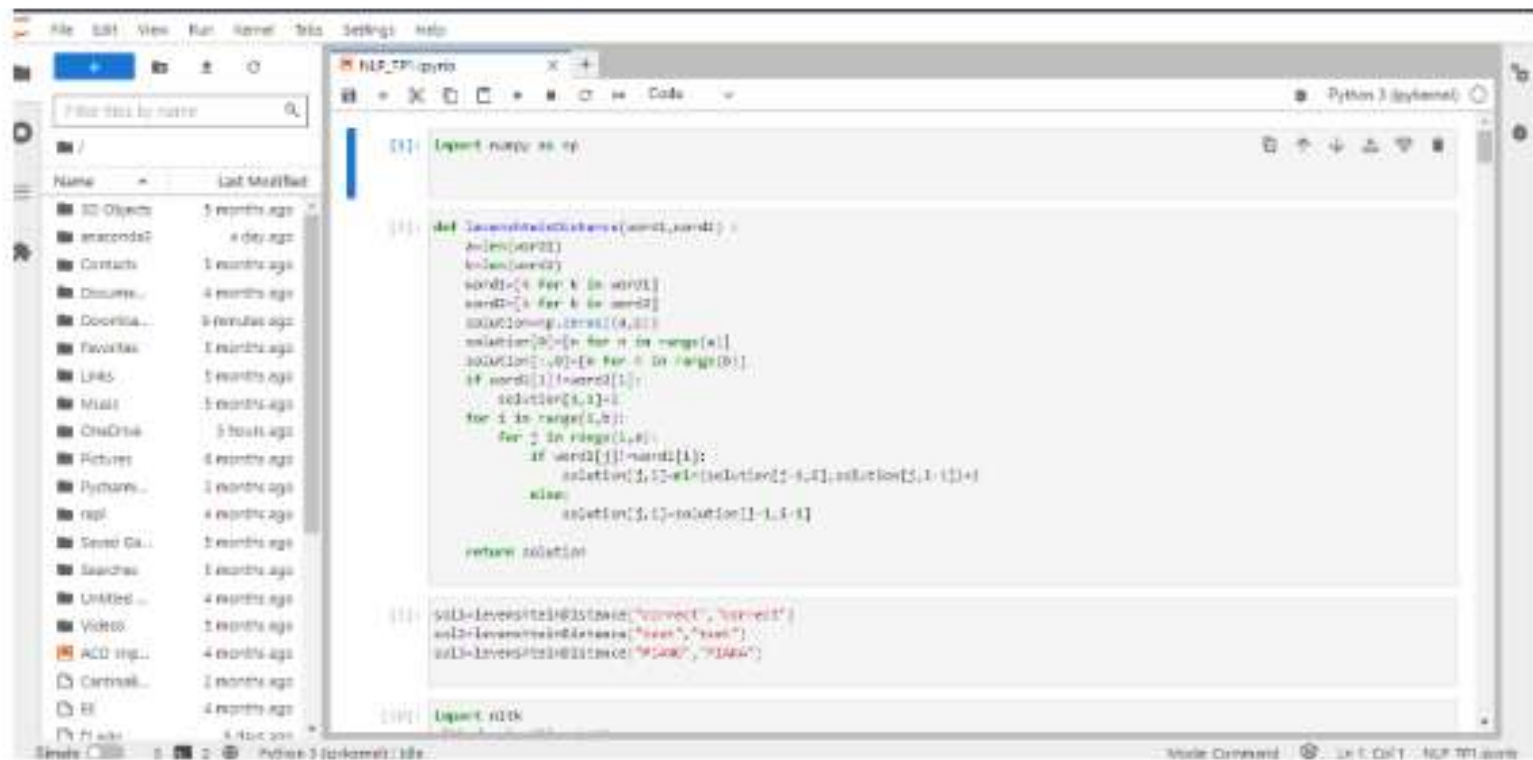
$$\text{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) \\ \min \begin{cases} \text{lev}_{a,b}(i-1,j) + 1 \\ \text{lev}_{a,b}(i,j-1) + 1 \\ \text{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} \end{cases}$$

where $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and 1 otherwise.



Exercise1

In this exercise, we will be implementing the Levenshtein distance function in Python code. We will write a function to calculate the distance between two given words. The implementation is shown in the following capture which will be explained in detail



```
[1]: Import numpy as np

[2]: def levenshteinDistance(word1, word2):
    len1=len(word1)
    len2=len(word2)
    word1=[0 for i in word1]
    word2=[0 for i in word2]
    solution=np.zeros((len1+1,len2+1))
    solution[0]=[i for i in range(len1+1)]
    solution[:,0]=[i for i in range(len2+1)]
    if word1[-1]==word2[-1]:
        solution[-1,-1]=1
    for i in range(1,len1):
        for j in range(1,len2):
            if word1[i]==word2[j]:
                solution[i,j]=min(solution[i-1,j],solution[i,j-1])+1
            else:
                solution[i,j]=min(solution[i-1,j],solution[i,j-1])+1
    return solution

[3]: sol1=levenshteinDistance('correct','correct')
sol2=levenshteinDistance('book','book')
sol3=levenshteinDistance('book','book')

[4]: Import nltk
```

We start by importing the numpy library then we start coding our function, the main idea is very simple, we create an empty matrix (word1.length * word2 length), the first row and first column are easy to fill (1 2 ...length). For the other matrix boxes there are two possible cases, if the letter of the first word is the same as the second one, the value is gonna be the same as the last one in the diagonal. If not, the value is the minimum around it plus one

		H	Y	U	N	D	A	I
	0	1	2	3	4	5	6	7
H	1	0	1	2	3	4	5	6
O	2	1	1	2	3	4	5	6
N	3	2	2	2	2	3	4	5
D	4	3	3	3	3	2	3	4
A	5	4	4	4	4	3	2	3

This logic can be simply implemented using for loops and if else condition as it shown in the capture

Exercise2

The pre-treatment and the normalisation of a text (NLTK and Spacy framework)



Text Pre-treatment and normalisation

The pretreatment and the normalisation of the input text remains one the most importance steps in the majority of NLP approaches as its garantie the algorithmic efficiency

There are many approaches and methods that are used in order to normalise a text and delete what may be considered as an unwanted information



NLTK

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning



Spacy

spaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython. It is a free and open-source library with a lot of built-in capabilities. It's becoming increasingly popular for processing and analyzing data in the field of NLP.

This second exercise is separated in two main parts. For the first one, we will explore some popular functionalities of the two open source platforms (NLTK and Spacy), then for the second part, we will apply what we will be through in a given text in order to achieve our last part of the report: the comparison between the two frameworks.

Exploring the Frameworks

Part1 : NLTK



Tokenizing



One of the first amazing functionalities of NLTK is tokenizing which may be expressed by separating text. There are two main forms of tokenizing that are presented in the attached capture



Parts of Speech Tagging (POS Tagging) is a procedure that marks up the words in text format for a specific segment of a speech based on its meaning and context. It is responsible for reading text in a language and assigning a distinct token to each word. It's also known as grammatical tagging.

In a lot of cases, it is conducive to finding out the grammatical classification of words. This process can always ameliorate the predictions of an NLP algorithm.

The pos tagging can easily be done using NLTK by calling the "pos_tag" function shown in the capture.

This function gives back the grammatical classification in observations. These lasts are listed in the following website which can be checked for more understanding of the output : "<https://www.guru99.com/pos-tagging-chunking-nltk.html>".



```
11: import nltk
12: sentence = nltk.chunk.ne_chunk(tagged)
13: print(sentence)

14:
15:
16:
17:
18:
19:
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:
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:
218
```


StopWords

```
[11]: from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words

[11]: ['a',
      'about',
      'above',
      'after',
      'again',
      'against',
      'all',
      'also',
      'am',
      'an',
      'and',
      'any',
      'are',
      'as',
      'at',
      'be',
      'because',
      'been',
      'but',
      'by',
      'can',
      'cannot',
      'could',
      'did',
      'do',
      'does',
      'each',
      'few',
      'for',
      'from',
      'further',
      'had',
      'has',
      'have',
      'he',
      'her',
      'his',
      'hundred',
      'into',
      'is',
      'it',
      'its',
      'just',
      'less',
      'like',
      'may',
      'me',
      'more',
      'most',
      'my',
      'no',
      'nor',
      'not',
      'of',
      'off',
      'on',
      'once',
      'only',
      'or',
      'other',
      'our',
      'out',
      'over',
      'that',
      'the',
      'their',
      'them',
      'then',
      'there',
      'these',
      'they',
      'this',
      'those',
      'through',
      'to',
      'too',
      'under',
      'until',
      'up',
      'us',
      'very',
      'was',
      'we',
      'were',
      'what',
      'when',
      'where',
      'which',
      'while',
      'who',
      'whom',
      'why',
      'with',
      'without',
      'would',
      'yet',
      'you',
      'your']
```

One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

°A stop word is a commonly used word (such as "the", "a", "an", or "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

In the picture shown above, we have listed all the stop Words for The English language. NLTK affords this functionality for a lot of languages such as French, Spanish, English and so on...Now we apply it to our example to see the result

```
NLP_TF.py
Python 3 (ipykernel)

('yourselves')

[10]: #tokens
filtered_sentence=[]
for w in tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
filtered_sentence

[10]: ['artificial',
      'intelligence',
      'considered',
      'needed',
      'looking',
      'first',
      'key',
      'look',
      'great',
      'inventions',
      '.',
      'it',
      'human',
      'and',
      'at',
      'major',
      '-']
```

StopWords

```
[11]: from nltk.corpus import stopwords
stop_words = stopwords.words('english')
stop_words

[11]: ['a',
      'about',
      'above',
      'after',
      'again',
      'against',
      'all',
      'also',
      'am',
      'an',
      'and',
      'any',
      'are',
      'as',
      'at',
      'be',
      'because',
      'been',
      'but',
      'by',
      'can',
      'cannot',
      'could',
      'de',
      'do',
      'does',
      'either',
      'else',
      'enough',
      'even',
      'ever',
      'except',
      'few',
      'for',
      'from',
      'further',
      'had',
      'has',
      'have',
      'he',
      'her',
      'hers',
      'him',
      'his',
      'how',
      'however',
      'i',
      'if',
      'in',
      'into',
      'is',
      'it',
      'its',
      'just',
      'keep',
      'last',
      'lastest',
      'latest',
      'less',
      'likely',
      'like',
      'likely',
      'may',
      'me',
      'might',
      'more',
      'most',
      'mostly',
      'much',
      'must',
      'my',
      'me',
      'no',
      'nor',
      'not',
      'of',
      'off',
      'often',
      'on',
      'once',
      'only',
      'or',
      'other',
      'ought',
      'over',
      'own',
      'same',
      'so',
      'some',
      'such',
      'than',
      'that',
      'the',
      'their',
      'them',
      'then',
      'there',
      'these',
      'they',
      'this',
      'those',
      'through',
      'to',
      'too',
      'toward',
      'towards',
      'under',
      'until',
      'up',
      'upon',
      'us',
      'very',
      'was',
      'we',
      'were',
      'what',
      'when',
      'where',
      'which',
      'while',
      'who',
      'whom',
      'why',
      'will',
      'with',
      'without',
      'would',
      'yet',
      'you',
      'your',
      'yourself']
```

One of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words.

°A stop word is a commonly used word (such as "the", "a", "an", or "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.

In the picture shown above, we have listed all the stop Words for The English language. NLTK affords this functionality for a lot of languages such as French, Spanish, English and so on...Now we apply it to our example to see the result

```
NLP_TF.py
Python 3 (ipykernel)

('yourselves')

[10]: #tokens
filtered_sentence=[]
for w in tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
filtered_sentence

[10]: ['artificial',
      'intelligence',
      'considered',
      'needed',
      'looking',
      'first',
      'key',
      'last',
      'great',
      'inventions',
      '.',
      'it',
      'human',
      'and',
      'at',
      'major',
      '-']
```

In this example , we created an empty list that will contains all the filtered words (That does not exist in the stop words list) , then we check on all the words of our text using a for loop and an if not condition.



```
NLP_TPS.pyth  Python 3.9.9
[1]: from nltk.stem import PorterStemmer
    ps=PorterStemmer()
    for i in tokens:
        print(ps.stem(i))

artifici
installig
is
conced
cloudy
as
an
interest
field
that
my
lead
to
can
great
invent
i
al
harder
are
al
major
```

The process of developing morphological variations of a root/base word is known as stemming. Stemming programs are sometimes known as stemming algorithms or stemmers. A stemming algorithm lowers the terms in a text to their root word.

However , we usually avoid to use stemming in the pre-treatment process as it may leads to a waste of information or it can give some meaningless or useless result as it is shown in the output of our example. There are some meaningless result like : "artifici" , "ana" . Despite , it can be vary helpful where we are interest in the classification of words primarily than their meaning.

Part2 : Spacy



spaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython

It is a free and open-source library with a lot of built-in capabilities. It's becoming increasingly popular for processing and analyzing data in the field of NLP.

In this part , we are going to explore some of the most known functionalities of the Spacy Open-Source. For the rest of this second part , we are going to work with the following example :

```
[1]: import spacy
nlp = spacy.blank("en")
test = nlp("Artificial intelligence is considered nowadays as an interesting feild that may leads to some great inventions , EL HARRARI ANAS Al Major 56%")
print(test.text)
```

Artificial intelligence is considered nowadays as an interesting feild that may leads to some great inventions , EL HARRARI ANAS Al Major 56%.

Tokenizing

```
[2]: for token in test:
    # Print the text tokens
    print(token.text)

Artificial
intelligence
is
considered
nowadays
as
an
interesting
feild
that
may
leads
to
some
great
inventions
,
EL
HARRARI
ANAS
Al
Major
56
%
```

Tokenizing using the Spacy Framework is one of the simplest tasks in text pre-treatment. Using "token. text" and a simple for loop gives u back all the text tokens; We can also get a specific word for the text or a slice from it, for example, I may get only the first word or the three last words of the text which is very helpful for text treatment. This manipulation is pretty much similar to string manipulation. This is shown in the following picture :

```
[1]: import spacy
nlp = spacy.blank("en")
test = nlp("Artificial intelligence is considered nowadays as an interesting feild that may leads to some great inventions , EL HARRARI ANAS Al Major 56%")
print(test.text)

Artificial intelligence is considered nowadays as an interesting feild that may leads to some great inventions , EL HARRARI ANAS Al Major 56%

[2]: first_token = test[0]
print(first_token.text)

Artificial

[3]: slice_text = test[0:4]
print(slice_text.text)

is considered
```


In fact , this simplicity allows us to have a great manipulation on our text which makes it easier to filter , select and even search for a specific information. For example , we want to verify if our text example contains a percentage and even get back the position and the value of this percentage. This can be done using just simple and ordinary lines of code

```
[21]: slice_text = text[2:4]
      print(slice_text)

Is contained

[22]: for token in test:
      # Check if the token resembles a number
      if token.like_num:
          # Get the next token in the document
          next_token = test[token.i + 1]
          # Check if the next token's text equals "%"
          if next_token.text == "%":
              print("Percentage found:", token.text)

Percentage found: 60.

[23]: !python -m spacy download en_core_web_sm

Collecting en-core-web-sm==0.0.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-0.0.0/en_core_web_sm-0.0.0-py3-none-any
```



```
en
%
-

[24]: # Iterate over the predicted entities
      for ent in doc.ents:
          # Print the entity text and its label
          print(ent.text, ent.label_)

[25]: for token in test:
      # Print the text tokens
      print(token.text)

artificial
```

We can iterate over the entities, the tagging and even the dependency labels the same way that we did with tokenizing by calling these attributes. This is shown in these captures above

Both NLTK and Spacy offer a lot of functionalities . In this manipulation , we were been trough some of the most popular and fundamental functions and attributes for the pre-treatment and the normalization of a text.

Comparison

In this final part , we will move to a very interesting part and a popular question in text pre-treatment : which is the best for this process (NLTK or Spacy)?

Overview

NLTK and Spacy are both one of the most popular Open-Sources for text pre-treatment and NLP in general. In fact, we may consider them the best for this kind of process

The main difference between NLTK and Spacy may remain the approach: NLTK consider every text as a string (This input is a string and so is the output) , whereas, Spacy uses the OOP approach where each text is an object and the outputs are objects too

NLTK	Spacy
The most popular NLP Library which offers a lot of functionalities	The fastest NLP Library
For each task, NLTK offers a lot of approaches	Easy to learn and use
Support a large number of languages	Provides built in words vectors

These may be considered as the main three advantages of each library which keep it popular and very helpful in NLP processing, in this part of comparison, we will also dress a table of the main drawbacks of each one :

NLTK	Spacy
Complexity of learning and use	Less flexible than NLTK
Quit slow comparing to SpacyThe c	Sentence Tokenization is very slow
Processes Strings (Does not use Ooed)	Only support 7 languages

To illustrate these differences, we are going to work with the following example (a text in french) and analyse the performance of each library

We choose a language supported by both NLTK and Spacy

Comparison

```
(1): import nltk
import spacy

nltk_text = """En semaine, je me lève à 6h30. Je prends une douche et un petit déjeuner et je pars au travail vers 7h15.
Pour arriver à mon entreprise à 8 heures, il me faut environ 45 minutes en voiture, mais parfois j'arrive en retard à cause des
vers 8 heures, je vais boire un café en salle de pause."""
print(nltk_text)

nlp = spacy.blank("fr")
spacy_text = nlp("""En semaine, je me lève à 6h30. Je prends une douche et un petit déjeuner et je pars au travail vers 7h15.
Pour arriver à mon entreprise à 8 heures, il me faut environ 45 minutes en voiture, mais parfois j'arrive en retard à cause des
vers 8 heures, je vais boire un café en salle de pause.""")
print(spacy_text.text)

En semaine, je me lève à 6h30. Je prends une douche et un petit déjeuner et je pars au travail vers 7h15.
Pour arriver à mon entreprise à 8 heures, il me faut environ 45 minutes en voiture, mais parfois j'arrive en retard à cause des
minutages.
```

Tokenizing

Word Tokenizing

```
[10]: import time
start_time = time.time()

tokens = nltk.word_tokenize(nltk_text)
print(tokens)

print("--- %s seconds ---" % (time.time() - start_time))

['In', 'matinée', '.', 'je', 'me', 'lève', 'à', '08h00', '.', 'je', 'prends', 'une', 'douche', 'et', 'un', 'petit', 'déjeuner',
'et', 'je', 'pars', 'au', 'travail', 'vers', '09h15', '.', 'Jusqu', 'à', '12h00', 'je', 'travaille', 'dans', 'mon', 'entreprise', 'à', 'Paris', 'heures', '.',
'Il', 'me', 'faut', 'arrêter', 'à', 'midi', 'pour', 'manger', 'et', 'boire', 'un', 'café', 'et', 'je', 'arrive', 'en', 'retard',
'à', 'cause', 'des', 'embouteillages', '.', 'Vers', '13h', 'heures', '.', 'Je', 'vais', 'boire', 'un', 'café', 'en', 'salle', 'de',
travail', '.']
--- 0.00389270494013867 seconds ---

[11]: start_time = time.time()

for token in spacy_text:
    print(token.text)

print("--- %s seconds ---" % (time.time() - start_time))

In
matinée
.
je
me
lève
à
08h00
.
je
prends
une
douche
et
un
petit
déjeuner
et
je
pars
au
travail
vers
09h15
.
Jusqu
à
12h00
je
travaille
dans
mon
entreprise
à
Paris
heures
.
Il
me
faut
arrêter
à
midi
pour
manger
et
boire
un
café
et
je
arrive
en
retard
à
cause
des
embouteillages
.
Vers
13h
heures
.
Je
vais
boire
un
café
en
salle
de
travail
.
--- 0.01797626495331128 seconds ---
```

Sentence tokenizing

```
[12]: start_time = time.time()

tokens = nltk.sent_tokenize(nltk_text)
print(tokens)

print("--- %s seconds ---" % (time.time() - start_time))

['In', 'matinée, je me lève à 08h00, je prends une douche et un petit déjeuner et je pars au travail vers 09h15. ', 'Jusqu', 'à', '12h00, je travaille dans mon entreprise à Paris heures. ', 'Il me faut arrêter à midi pour manger et boire un café, et je arrive en retard à cause des embouteillages. ', 'Vers 13 heures, je vais boire un café en salle de travail. ']
--- 0.00389270494013867 seconds ---

[13]: start_time = time.time()

nlp = spacy.load("en_core_web_sm")
for sent in spacy_text.sents:
    print(sent.text)

print("--- %s seconds ---" % (time.time() - start_time))

In
matinée, je me lève à 08h00, je prends une douche et un petit déjeuner et je pars au travail vers 09h15.

Jusqu
à
12h00, je travaille dans mon entreprise à Paris heures, Il me faut arrêter à midi pour manger et boire un café, et je arrive en retard à cause des embouteillages.

Vers 13 heures, je vais boire un café en salle de travail.
--- 1.40355326541793 seconds ---
```


Conclusion

The pre-treatment of a given text remains an essential part of any other process. This work includes tokenizing, filtering, tagging, classifying and normalizing the text and its corpora

One of the fundamental tools to achieve this pre_treatment is the two popular Open-Sources Spacy and NLTK

While both can theoretically accomplish any NLP task, each one excels in certain scenarios

Both NLTK and spaCy offer great options when you need to build an NLP system. As we have seen, however, spaCy is the right tool to use in a production environment. Its underlying philosophy – providing a service rather than being a tool – is behind its extreme user-friendliness and performance. spaCy just gets the job done!