phrase_model

September 8, 2019

[1]:
```
from phrase_mt import PhraseModel
from collections import defaultdict
import nltk
```

[3]:
```
def read_pre_aligns(st_filename1, st_filename2, ts_filename1, ts_filename2):
    st_t_file = open(st_filename1, 'r')
    st_s_file = open(st_filename2, 'r')

    st_all_aligns = []

    I = 0
    for sen_s, sen_t in zip(st_t_file.readlines(), st_s_file.readlines()):
        if I < 250000:
            sen_aligns = defaultdict(lambda: str)
            for word_s, word_t in zip(sen_s.strip('\n').split(), sen_t.
 ↪strip('\n').split()):
                sen_aligns[word_s] = word_t
            st_all_aligns.append(sen_aligns)
        I += 1


    ts_t_file = open(ts_filename1, 'r')
    ts_s_file = open(ts_filename2, 'r')

    ts_all_aligns = []

    I = 0
    for sen_s, sen_t in zip(ts_t_file.readlines(), ts_s_file.readlines()):
        if I < 250000:
            sen_aligns = defaultdict(lambda: str)
            for word_s, word_t in zip(sen_s.strip('\n').split(), sen_t.
 ↪strip('\n').split()):
                sen_aligns[word_s] = word_t
            ts_all_aligns.append(sen_aligns)
        I += 1
    return st_all_aligns, ts_all_aligns

st_aligns, ts_aligns = read_pre_aligns("st_aligns_s.txt", "st_aligns_t.txt",␣
↪"ts_aligns_s.txt", "ts_aligns_t.txt")
```

[56]:
```
st_aligns[2324]
```

[56]:
```
defaultdict(<function __main__.read_pre_aligns.<locals>.<lambda>()>,
            {'CIBLER': 'TARGET',
             'PRESTATIONS': 'TO',
             'DE': 'TO',
             '.': 'TO',
             'LES': 'IT',
             'IL': 'IS',
             'ESSAIE': 'IT'})
```

[57]:
```
ts_aligns[2324]
```

[57]:
```
defaultdict(<function __main__.read_pre_aligns.<locals>.<lambda>()>,
            {'TO': 'DE',
             'IS': 'DE',
             'TRYING': 'DE',
             'IT': 'IL',
             'TARGET': 'CIBLER'})
```

[6]:
```
len(st_aligns)
```

[6]:
```
207688
```

[7]:
```
ts_aligns[0]
```

[7]:
```
defaultdict(<function __main__.read_pre_aligns.<locals>.<lambda>()>,
            {'HANSARD': 'HANSARD',
             'DEBATES': 'DÉBATS',
             'SENATE': 'SÉNAT',
             'OF': 'DU',
             ')': ')',
             '(': '(',
             'THE': 'DU'})
```

[8]:
```
P_MT = PhraseModel(st_aligns, ts_aligns, 'hansards.36.ca.f.tok', 'hansards.36.
↪ca.e.tok')
```

[9]:
```
len(P_MT.s_sens)
```

[9]:
```
207688
```

[16]:
```
P_MT.grow_alignments()
```

```
0
10000
20000
```

```
30000
40000
50000
60000
70000
80000
90000
100000
110000
120000
130000
140000
150000
160000
170000
180000
190000
200000
```

[19]:
```
P_MT.sen_overlaps_ts[125]
```

[19]:
```
defaultdict(<function
phrase_mt.PhraseModel.get_align_overlaps.<locals>.<lambda>()>,
            {'BY': ['PAR'],
             'BESTOWS': ['CONFÈRE'],
             'FOR': ['À'],
             'VALUES': ['VALEURS'],
             'OF': ['DU', 'COMMUNES'],
             'LEGITIMACY': ['LÉGITIMITÉ'],
             'CANADA': ['CANADA'],
             'ALL': ['TOUS'],
             'WITH': ['AVEC'],
             ',': [',', 'TANT'],
             '.': ['.'],
             'GOVERNMENT': ['GOUVERNEMENT'],
             'ELECTED': ['ÉLU'],
             'THE': ["L'", 'LA', 'LE', 'AU'],
             'THAT': ['QUE', 'CELA', 'LUI'],
             'CANADIANS': ['LES', 'CANADIENS'],
             'AND': ['LES'],
             'ENDOWED': ['CANADIENS'],
             'THIS': ['CELA'],
             'WILL': [',', 'DÉFENDRA'],
             'UP': ['À'],
             'SHARED': ['LES'],
             'HOME': ['ADHÈRENT'],
             'STAND': ['À'],
             'AT': ['AUXQUELLES']})
```

[20]:
```
P_MT.sen_overlaps_st[125]
```

[20]:
```
defaultdict(<function
phrase_mt.PhraseModel.get_align_overlaps.<locals>.<lambda>()>,
            {'PAR': ['BY'],
             'CONFÈRE': ['BESTOWS'],
             'À': ['FOR', 'UP', 'STAND'],
             'VALEURS': ['VALUES'],
             'DU': ['OF'],
             'LÉGITIMITÉ': ['LEGITIMACY'],
             'CANADA': ['CANADA'],
             'TOUS': ['ALL'],
             'AVEC': ['WITH'],
             ',': [',', 'WILL'],
             '.': ['.'],
             'GOUVERNEMENT': ['GOVERNMENT'],
             'ÉLU': ['ELECTED'],
             "L'": ['THE'],
             'QUE': ['THAT'],
             'LES': ['CANADIANS', 'AND', 'SHARED'],
             'CANADIENS': ['CANADIANS', 'ENDOWED'],
             'LA': ['THE'],
             'CELA': ['THAT', 'THIS'],
             'LUI': ['THAT'],
             'LE': ['THE'],
             'TANT': [','],
             'AU': ['THE'],
             'DÉFENDRA': ['WILL'],
             'COMMUNES': ['OF'],
             'ADHÈRENT': ['HOME'],
             'AUXQUELLES': ['AT']})
```

[21]:
```
len(P_MT.sen_overlaps_st)
```

[21]:
```
207688
```

[22]:
```
P_MT.sen_overlaps_st[0]
```

[22]:
```
defaultdict(<function
phrase_mt.PhraseModel.get_align_overlaps.<locals>.<lambda>()>,
            {'SÉNAT': ['SENATE'],
             'HANSARD': ['HANSARD'],
             ')': [')'],
             '(': ['('],
             'DÉBATS': ['DEBATES'],
             'DU': ['OF', 'THE']})
```

[23]:
```
P_MT.sen_overlaps_ts[0]
```

[23]:
```
defaultdict(<function
phrase_mt.PhraseModel.get_align_overlaps.<locals>.<lambda>()>,
            {'SENATE': ['SÉNAT'],
             'HANSARD': ['HANSARD'],
             ')': [')'],
             '(': ['('],
             'DEBATES': ['DÉBATS'],
             'OF': ['DU'],
             'THE': ['DU']})
```

[24]:
```
def save_aligns(st_all_aligns, ts_all_aligns):

    ts_s_file = open("final_aligns_ts_s.txt", 'w')
    ts_t_file = open("final_aligns_ts_t.txt", 'w')

    for sen_table in ts_all_aligns:
        for word_s, word_list in sen_table.items():
            ts_s_file.write(word_s + ' ')
            ts_t_file.write(' '.join(word_list) + ' ') if word_list != None␣
 ↪else 'None '
        ts_s_file.write('\n')
        ts_t_file.write('\n')


    st_s_file = open("final_aligns_st_s.txt", 'w')
    st_t_file = open("final_aligns_st_t.txt", 'w')

    for sen_table in st_all_aligns:
        for word_s, word_list in sen_table.items():
            st_s_file.write(word_s + ' ')
            st_t_file.write(' '.join(word_list) + ' ') if word_list != None␣
 ↪else 'None '
        st_s_file.write('\n')
        st_t_file.write('\n')
```

[25]:
```
save_aligns(P_MT.sen_overlaps_st, P_MT.sen_overlaps_ts)
```

[26]:
```
phrase_lex = P_MT.get_phrase_lex()
```

```
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
```

```
10000
11000
12000
13000
14000
15000
16000
17000
18000
19000
20000
21000
22000
23000
24000
25000
26000
27000
28000
29000
30000
31000
32000
33000
34000
35000
36000
37000
38000
39000
40000
41000
42000
43000
44000
45000
46000
47000
48000
49000
50000
51000
52000
53000
54000
55000
56000
57000
```

58000
59000
60000
61000
62000
63000
64000
65000
66000
67000
68000
69000
70000
71000
72000
73000
74000
75000
76000
77000
78000
79000
80000
81000
82000
83000
84000
85000
86000
87000
88000
89000
90000
91000
92000
93000
94000
95000
96000
97000
98000
99000
100000
101000
102000
103000
104000
105000

```
106000
107000
108000
109000
110000
111000
112000
113000
114000
115000
116000
117000
118000
119000
120000
121000
122000
123000
124000
125000
126000
127000
128000
129000
130000
131000
132000
133000
134000
135000
136000
137000
138000
139000
140000
141000
142000
143000
144000
145000
146000
147000
148000
149000
150000
151000
152000
153000
```

```
154000
155000
156000
157000
158000
159000
160000
161000
162000
163000
164000
165000
166000
167000
168000
169000
170000
171000
172000
173000
174000
175000
176000
177000
178000
179000
180000
181000
182000
183000
184000
185000
186000
187000
188000
189000
190000
191000
192000
193000
194000
195000
196000
197000
198000
199000
200000
201000
```

```
202000
203000
204000
205000
206000
207000
```

[28]:
```
P_MT.phrase_lex_to_probs()
```

[95]:
```
P_MT.phrase_lex[('DIXIÈME',)]
```

[95]:
```
defaultdict(<function phrase_mt.PhraseModel.phrase_lex_to_probs.<locals>.<lambda
>.<locals>.<lambda>()>,
            {('REPORT',): -1.845826690498331,
             ('TENTH',): -0.5465437063680699,
             ('OF',): -1.55814461804655})
```

[31]:
```
P_MT.build_trigram_model()
```

[149]:
```
decode_sentence(P_MT, 'JE CROIS SAVOIR QUE LA PRATIQUE VEUT QUE LORSQUE SON␣
 ↪EXCELLENCE LA GOUVERNEURE GÉNÉRALE VIENT AU SÉNAT POUR DONNER LA SANCTION␣
 ↪ROYALE , LE PREMIER MINISTRE SOIT PRÉSENT .'.upper())
```

[149]:
```
[(['I',
   'UNDERSTAND',
   'THAT',
   'THE',
   'PRACTICE',
   'WANTS',
   'WHEN',
   'HIS',
   'EXCELLENCY',
   'GOVERNOR',
   'GENERAL',
   'JUST',
   'TO',
   'THE',
   'SENATE',
   'TO',
   'GIVE',
   'ASSENT',
   'ROYAL',
   ',',
   'THE',
   'PRIME',
   'MINISTER',
   'BE',
   'PRESENT',
   '.'],
  -1467.3620565460023),
 (['I',
   'UNDERSTAND',
   'THAT',
   'THE',
   'PRACTICE',
   'WANTS',
   'WHEN',
   'HIS',
   'EXCELLENCY',
   'GOVERNOR',
   'GENERAL',
   'JUST',
   'TO',
   'THE',
   'SENATE',
   'TO',
   'GIVE',
   'ASSENT',
   'ROYAL',
   ',',
   'THE',
   'PRIME',
   'MINISTER',
   'BE',
   'PRESENT',
   '.'],
  -1438.1370709811415)]
```

[189]:
```
ref = "IT IS MY UNDERSTANDING THAT IT IS THE PRACTICE OF THIS PLACE THAT , WHEN␣
 ↪HER EXCELLENCY THE GOVERNOR GENERAL COMES TO THE SENATE FOR ROYAL ASSENT ,␣
 ↪THE FIRST MINISTER , THE PRIME MINISTER , WILL BE PRESENT .".lower()
```

[190]:
```
can =  "I UNDERSTAND THAT THE PRACTICE WANTS WHEN HIS EXCELLENCY GOVERNOR␣
 ↪GENERAL JUST TO THE SENATE TO GIVE ASSENT ROYAL THE PRIME MINISTER BE␣
 ↪PRESENT".lower()
```

[191]:
```
nltk.translate.bleu_score.sentence_bleu([ref], can)
```

[191]:
```
0.531366506973441
```