Programmieraufgabe 2

Ekatherina Tikhoncheva

8. Juli 2014

1 Der Basis-Decoder

Das kommentierte Skript **decode**, so wie es gegeben wurde, ist unter den Namen **decodeCommented** gespeichert.
Das beste Ergebnis (LM+TM) bei der Stackgröße 1, das man mit diesem Skript erzielen kann, ist -1442.727775. Wenn man die Stackgröße auf 1000 erhöht, ist das Ergebnis (LM+TM) besser geworden : -1439.213923.

2 Decoding mit Phrasenvertauschung

Wie man gesehen hat, behält der Basis Decoder die Reihenfolge der französischen Phrasen bei. Wir möchten das ändern, so dass der neue Decoder benachbarte Phrasen vertauschen kann.

Um die Implementierungsidee zu erklären betrachten wir ein kleines Beispiel. Sei der französischen Satz *un Comité de sélection a été constitué* gegeben.
Wie davor gehen wir monoton von links nach rechts und suchen nach möglichen Phrasen:

1. Phrase „un“ besteht aus ein Wort, ist in TM ⇒ übersetze die Phrase und shau nach den Phrasen danach.
Erste Phrase nach „un“ ist „Comité“ ⇒ vertausche die Phrasen und übersetze jede einzige vertauschte Phrase separat.
Zweite Phrase nach „un“ ist „Comité de“ ⇒ vertausche die Phrasenund übersetze jede einzige vertauschte Phrase separat.

2. Phrase „un Comité“ besteht aus zwei Wörtern, ist in TM ⇒ übersetze und shau nach den Phrasen danach.
Erste Phrase danach ist „de“ ⇒ vertausche die Phrasen und übersetze jede einzige vertauschte Phrase separat.
usw.

For Implementation siehe Datei **decodeMy**. Hier ist ein Ausschnitt davon mit den meisten Änderungen.
Um nach einer gefundenen Phrase alle mögliche Phrasen danach zu finden, brauchen wir eine zusätzliche Schleife (Zeile 18). Die Schleife ueber $swap = 1, 2$ unterscheidet ob es sich um eine einzige Phrase oder um Vertauschung von zwei Phrasen handelt (Zeilen 20, 23 und 56).
Wenn wir zwei nacheinandre folgende Phrasen P_1 und P_2 gefunden haben, dann werden in

entsprechenden Stack alle Kombinationen von möglichen Übersetzungen der Phrase P_2 gefolgt von allen möglichen Übersetzungen der Phrase P_1 (Zeile 74 – 115).

decodeMyCut

```
1    # for each stack
2    for i, stack in enumerate(stacks[:-1]):
3      #
4      #sys.stdout.write("\nStack")
5      #sys.stdout.write(str(i))
6      #sys.stdout.write("\n")
7      #sys.stdout.write(str(stack))
8
9      # sort translation hypothesis from the current stack in decreasing order of logarithmic probabiliti
10     # and consider first s best elements (pruning with the stack size s, default s=1)
11     for h in sorted(stack.itervalues(),key=lambda h: -h.logprob)[:opts.s]:
12       # NEW : save current coverage of the hypothesis h
13       hcoverage = h.coverage
14
15       # for each new hypothesis that can be derived from h
16       for j in xrange(i,len(f)+1):
17         # NEW  + additional loop to run over all posible phrase after already found first one
18         for k in xrange(j+1,len(f)+1):
19           # NEW swap indicates swaping of phrases
20           for swap in xrange(0,2):
21
22             # NEW swap=0 means add one phrase how it is
23             if swap==0:
24               coverage = getcoverage(i,k) # save coverage of the phrase
25               stackNum  = k # stack number where to put new translation
26               fphrase = f[i:k]  # phrase
27
28               # if TM knows the phrase and there is no overlap in coverage vector
29               if (fphrase in tm) and (hcoverage & coverage == 0) and k-j>swap:
30                 # we add we before all possible translations into corresponding stacks
31                 for phrase in tm[fphrase]:
32                   # recalculate probability of the hypothesis extended with the new phrase
33                   logprob = h.logprob + phrase.logprob
34                   # save current state of the translation ()
35                   lm_state = h.lm_state
36                   # for each english word in the phrase
37
38                   for word in phrase.english.split():
39                    # calculate a probability based on lm
40                     (lm_state, word_logprob) = lm.score(lm_state, word)
41                     logprob += word_logprob
42                   #end for loop word
43
44                   # add a probability of the last word, if we reached the end of the foreign sentence
45                   logprob += lm.end(lm_state) if j == len(f) else 0.0
46                   # create new hypothesis
47                   # NEW calculate a new coverage vector
48                   new_hypothesis = hypothesis(logprob, lm_state, h, phrase, hcoverage | coverage)
49                   # expanded hypothesis is placed in corresponding stack
50                   if lm_state not in stacks[stackNum] or stacks[stackNum][lm_state].logprob < logprob:
51                     stacks[stackNum][lm_state] = new_hypothesis
52                 # end loop over phrases
53             # end if swap==0
54
55             # NEW swap=1 means : find two phrases one after another and try to swap them
56             if swap==1 and k>j+1: # second case because we dont want to swap phrase with it self
57
58               coverage = getcoverage(i,k)       # save coverage of the phrase
59               stackNum  = k                     # stack number where to put new translation
60
61               # split long phrase that we found in two an look if they build two meaningfull phrases
62               fphrase1 = f[i:j+1]
63               fphrase2 = f[j+1:k]
```

```
64                if (fphrase1 in tm and fphrase2 in tm):
65
66                  # if TM knew both as phrases we can swap them
67                  fphrase = fphrase2+fphrase1
68
69                  # now add fphrase2 first with all possible translations
70                  # and then direckt phrase1 with all possible translations
71                  for phrase2 in tm[fphrase2]:
72
73                    # recalculate probability of the hypothesis extended with the new phrase
74                    logprob = h.logprob + phrase2.logprob
75                    # save current state of the translation ()
76                    lm_state = h.lm_state
77                    # for each english word in the phrase
78
79                    for word in phrase2.english.split():
80                      # calculate a probability based on lm
81                      (lm_state, word_logprob) = lm.score(lm_state, word)
82                      logprob += word_logprob
83                    #end for loop word
84
85                    # add a probability of the last word, if we reached the end of the foreign sentence
86                    logprob += lm.end(lm_state) if j == len(f) else 0.0
87
88                    # create new hypothesis
89                    new_hypothesis1 = hypothesis(logprob, lm_state, h, phrase2, hcoverage | (coverage/2))
90
91                    # phrase2 is saved we extand it with all possible traslations for phrase1
92                    for phrase1 in tm[fphrase1]:
93
94                      # recalculate probability of the hypothesis extended with the new phrase
95                      logprob = new_hypothesis1.logprob + phrase1.logprob
96                      # save current state of the translation ()
97
98                      # NEW actuall state is previous hypothesis new_hypothesis1
99                      lm_state = new_hypothesis1.lm_state
100                     # for each english word in the phrase
101
102                     for word in phrase1.english.split():
103                       # calculate a probability based on lm
104                       (lm_state, word_logprob) = lm.score(lm_state, word)
105                       logprob += word_logprob
106                     #end for loop word
107
108                     # add a probability of the last word, if we reached the end of the foreign sentence
109                     logprob += lm.end(lm_state) if j == len(f) else 0.0
110
111                     # create new hypothesis
112                     new_hypothesis2 = hypothesis(logprob, lm_state, new_hypothesis1, phrase1, hcoverage
113                     # expanded hypothesis is placed in corresponding stack
114                     if lm_state not in stacks[stackNum] or stacks[stackNum][lm_state].logprob < logprob
115                         stacks[stackNum][lm_state] = new_hypothesis2
116                   # end loop over phrases1
117
118                 # end loop over phrases2
119
120               # end if fphrase1 in tm and fphrase2 in tm
121             # end if swap==1
122
123           # end swap
124         #end loop k
125       #end loop j
126
127     # end loop over hypothesis in the current stack
128   # end loop over stacks
```

Die Ergebnisse dieses Decoder sind :

-1409.200122 für Stackgröße 1 (Basis-Decoder -1442.727775)
-1391.747648 für Stackgröße 1000 (Basis-Decoder -1439.213923)

Bei dem Versuch die Pruning Konstante s hoehe zu wählen, hat sich das Ergebnis nicht verbessert. -1391.747648 für Stackgröße 5000 (Basis-Decoder -1439.213923).