Dsp hw3

黃郁庭 r05922038

1. **My environment**

   CSIE workstation, ex. Linux9

   OS version : Linux linux9 4.12.3-1-ARCH #1 SMP PREEMPT Sat Jul 22 15:32:02 UTC 2017 x86_64 GNU/Linux

   ***pyrhon2* version** : Python 2.7.14 (dsp_hw3_r05922038/mapping.py 只能用 python 2.7 執行，若用 python 3 會出錯，Makefile 中已將 python 改為 python2，固可直接 type "*make map*" 於 CSIE workstation 執行)。

   ***g++* version** : gcc version 7.2.0 (GCC)

2. **How to compile my program**

   可以用 dsp_hw3_r05922038/Makefile：type "make all" 即會編譯 dsp_hw3_r05922038/mydisambig.cpp 出可執行檔 mydisambig。若要設定 MACHINE_TYPE 或 SRIPATH (指向 srilm-1.5.10)，可以 type "make MACHINE_TYPE=i686-m64 SRIPATH=path_to_rilm-1.5.10 all"。

3. **How to execute my program**

   a. **mapping.py**：ZhuYin-Big5.map

      除了可以用 Makefile 執行 mapping.py 將 Big5-ZhuYin.map 轉成 ZhuYin-Big5.map (type "*make map*")，也可以 type "*python2 mapping.py Big5-ZhuYin.map ZhuYin-Big5.map*"產生 ZhuYin-Big5.map。(這裡只能用 Python 2.7 的版本，若用 python 3 會出錯)

   b. **mydisambig.cpp**：reseult2/1.txt~10.txt

      bigram

      除了可以用 Makefile 執行 mydisambig 將 testdata/1.txt~10.txt 生成 reseult2/1.txt~10.txt (type "make run"，可以如同上述 2.設定 MACHINE_TYPE 或 SRIPATH)，也可以 type "*./mydisambig -text testdata/1.txt -map ZhuYin-Big5.map -lm bigram.lm -order 2 > reseult2/1.txt*"，其中 *1.txt* 要依序改到 *10.txt* (也就是 *reseult2/1.txt* 產生後再將上述指令 1.txt 的地方都改成 2.txt，並依此步驟改下去，便能產生 reseult2/1.txt~10.txt)。

      trigram (寫好久 QQ)

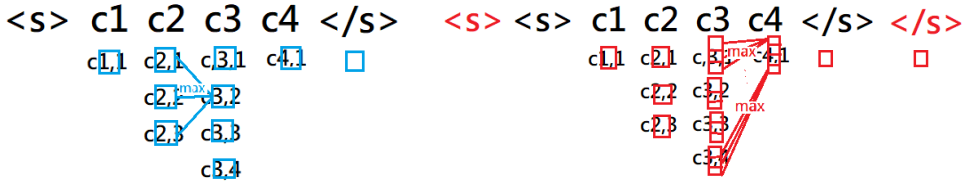
      除了可以 type "make LM=trigram.lm run"用 trigram 來產生結果(產生的 1.txt~10.txt 會存在 *reseult2* 中)，也可以 type "*./mydisambig -text testdata/1.txt -map ZhuYin-Big5.map -lm trigram.lm -order 3 > reseult2/1.txt*"， 其中 *1.txt* 也要如同上述依序改到 *10.txt*。

4. **What I have done**

SRILM disambig

我先用 separator_big5.pl 將 testdata 中的 raw data 1.txt~10.txt 切成一個字一個字(以空格區分)，再用這些 data 訓練 bigram language model (用 SRILM 的 ngram-count 程式)，然後再用 SRILM 的 disambig 程式生成 reseult1/1.txt~10.txt。

Mydisambig-bigram



以左圖中一個 sentence “<s> c1 c2 c3 c4 </s>為例，若 ci 是一個中文字，則在 ci 的位置只有一個 state，如 c1 與 c4；若 ci 是一個注音，則在 ci 的位置會有 ni 個 state，每個 state 是 ZhuYin-Big5.map 中 ci 那一列對應到的中文字(共有 ni 個)。用 viterbi 演算法計算 sentence 中那些注音應該代換成哪個中文時，當計算到 ci 時，每個 ci,j 會找出 c(i-1),j’ for all j’中轉換到 ci,j 最大的機率當作 ci,j 的 accumulated probability，並記錄此 state j’ (以 c3,2：為例 accumulated prob. of c3,2 = max j {(accumulated prob. of c2,j)*(transition prob. from c2,j to c3,2)} ，previous state of c3,2 = argmax j {(accumulated prob. of c2,j)*(transition prob. from c2,j to c3,2)}。在最後輸出句子時只須依循所記錄的 previous state 從</s>往前 index 便可輸出最後結果。在 viterbi 演算法中每個 iteration 只需 maintain 一維的資料，此即 dynamic programming。

Mydisambig-trigram

以右圖為例，用 viterbi 演算法計算 sentence 中那些注音應該代換成哪個中文時，當計算到 ci 時，每個 ci,j 會記錄 n 個 accumulated prob. (來自 i-2 與 i-1)與 n 個 state id (來自 i-2)。以 c4 為例，c4,1 會分別記錄來自 c3,1~c3,4 的 accumulated prob.，假設紀錄 c3,1 的為 c4,1,1，若在 c3,1,1~c3,1,3 中到達 c4,1,1 的最大機率為 c3,1,2 的 accumulated prob.乘上 transition prob. P(c4,1|c2,2 c3,1)，則 c4,1,1 除了要記錄此最大機率為自己的 accumulated prob.，還要記錄此機率來自 c2 的 state 2。

因為在 viterbi 演算法中每個 iteration 需 maintain 二維的資料，所以計算複雜度較 bigram 高，為了要在規定的 10 min 內 run 完 1.txt~10.txt，我在每個 state 只保存前三個最大的機率。

比較 bigram 與 trigram 的結果可以發現 trigram 幾乎表現得比 bifram 好，尤其是在以三個字或四個字組成的詞較多的句子中可以發現。例如在 1.txt 中，bigram 會將”求好心く”翻成”求好心情”，而 trigram 翻成”求好心切”較合文意；或是 bigram 會將”ㄆ灣族”翻成”波灣族”，而 trigram 則可以正確翻成”排灣族”。