Machine Discovery HW1_2 Report b02902030 曹燁文

HMM Learning and Testing
Use Baum-Welch Algorithm
Reference:
Wikipedia
https://en.wikipedia.org/wiki/Baum%E2%80%93Welch_algorithm
數位語音處理概論 投影片
http://speech.ee.ntu.edu.tw/DSP2016Autumn/

根據Baum-Welch演算法，先初始化a, b, pi值，即bigram, encode, init的值，在這次作業我使用uniform分佈的固定值初始化。每次iteration分別算出alpha, beta, gamma, epsilon的值，然後使用這些數值更新a, b, pi。
alpha和beta分別為Forward Algorithm和Backward Algorithm，計算觀察到1~t的觀察值之後在時間t位在state i的機率，以及給定在時間t位在state i，得到t+1~T的觀察值的機率。根據這兩項數值，可以算出gamma和epsilon，也就是給定所有觀察值，在t時間的state為i的機率，以及給定所有觀察值，在t時間state為i和t+1時間state為j的機率。然後計算所有時間t的epsilon及gamma值總和，根據這兩個值更新三項初始參數。
另外，為了檢查是否正確，所以初始化固定是uniform分佈，有嘗試過幾次random init，但效果沒有uniform好。

速度優化：
（以下秒數皆為在valid上測試的結果）一個iteration大約130秒，使用openmp平行化之後大概可以加速到43秒。然而跑test的時候發現過於緩慢以及所需記憶體過大的問題，所以改變寫法，不紀錄所有的gamma和epsilon值，而是直接紀錄數值的和，減少大量的記憶體使用，就算不使用平行也加速到大約70秒/iteration。
之後根據：b02705008 江昱熹 同學教我的方法，使程式能跳過機率確定是0的部份，讓整體速度大幅提昇，一個iteration大約15秒。方法如下：首先建立一個vector，叫做encode，每一個encode[j]裡面存的是哪些b[i][j]非零，所以更新參數時只要計算這些index內的即可。加上這幾項變數的計算都有遞迴的性質在，所以除了encode是零的地方，alpha, beta, gamma, epsilon 計算時也都能再次過濾掉是零的部份，讓速度再次提昇。Viterbi Algorithm也可以應用這項技巧，減少許多運算時間。
最後是我發現計算gamma和epsilon的時候，原本的公式都要除一項P(O|lambda)，但是在更新a和b的時候發現這個值會被約掉，也就是說根本不用計算。減少這部份的計算之後，讓速度提升到一個iteration大約7~8秒，尤其是gamma和epsilon的部份加速了大約8倍。

停止條件：
我嘗試過計算兩次iteration之間encode和bigram機率的RMSE值、兩次iteration之間result差多少個等方法，也嘗試過累積50個iteration的結果然後投票選出result，希望能從中看出如何讓accuracy最高，但是發現沒有這些結果沒有明顯的相關。valid很早就

開始收斂，大約25次iteration之後accuracy會明顯降低，而valid2則是跑到50個iteration的時候還有明顯上升的趨勢。於是最終test1,test2選擇跑50個iteration，也許還沒收斂，但是從valid的經驗來看就算跑過頭了但是accuracy也不會降低太多。

Trigram:
這部份有嘗試實作(trigram_bug.cpp)，但由於時間不足所以沒有完整寫出來。想法如下：在bigram的想法底下是看一個state轉移到另一個state，所以在trigram底下就變成看兩個state轉移到一個state，而encode可以看成是看兩個state轉移到兩個state，所有計算過程需要的alpha, beta等變數都需乘上state倍，所以計算過程中有遇到所需記憶體太大的問題，也因為需要計算的state變多，計算時間也大幅提昇。