r07944037 陳耀新

這個作業需要以C / C++來完成, 整體而言雖然說不上難寫, 不過還是花了不少力氣在這上面.
實作過程中會用到像是 pointer / malloc 還有許多 C 的 function等等, 這些東西讓我打開了塵封已久的記憶, 下面會附上一些 source code 的截圖, 及簡略地說明.

training

```c
#include "hmm.h"
#include <time.h>

int main(int argc, char *argv[]) {

    char *p;
    long conv = strtol(argv[1], &p, 10);
    int iteration = conv;
    char *model_init = argv[2];
    char *train_seq = argv[3];
    char *model = argv[4]; // output
    long trainingTime = CLOCKS_PER_SEC * 60;
    HMM hmm_initial;

    loadHMM( &hmm_initial, model_init );
    dumpHMM( stderr, &hmm_initial );
    printf("============================\n");
    printf("System Running...\n");
    printf("training HMM...\n");
    printf("============================\n");
    int count;
    while((clock()/CLOCKS_PER_SEC < 60) && count < iteration)
    {
        hmm_initial = *(trainingModel(train_seq, &hmm_initial));
        count++;
    }
    FILE *fp = open_or_die( model, "w");
    dumpHMM( fp, &hmm_initial);
    fclose(fp);
    printf("DONE!\n");
    return 0;
}
```

```c
// printf("re-estimate all parameters for lamda...\n");
for(int i = 0; i < hmm_initial->state_num; i++) {
   hmm_initial->initial[i] = pi_sum[i] / lines;
}//printf("end_of_pi\n");

for(int i = 0; i < hmm_initial->state_num; i++) {
   for(int j = 0; j < hmm_initial->state_num; j++) {
      hmm_initial->transition[i][j] = epsilon_sum[i][j] / gama_sum_A[i];
   }
}//printf("end_of_A\n");

for(int i = 0; i < hmm_initial->observ_num; i++) {
   for(int j = 0; j < hmm_initial->state_num; j++) {
      hmm_initial->observation[i][j] = observation_sum[i][j] / gama_sum_B[j];
   }
}//printf("end_of_B\n");
```

```c
alpha = calculateAlpha(hmm_initial, token, T);
beta = calculateBeta(hmm_initial, token, T);
gama = calculateGama(alpha, beta, hmm_initial, T);
epsilon = calculateEpsilon(alpha, beta, hmm_initial, token, T);
```

這裡用到了 hmm.h 及 time.h
在 hmm.h 中會使用原本附上的 loadHMM 來存取 model_init.txt 的資料
hmm.h的 trainMode 則是主要的 training過程, time.h用於系統的 time control

trainModel每次會讀一列sequence, 並且計算其
alpha / beta / gama / epsilon矩陣, 而re-estimate會需要額外累加各個 matrix sum, 利用這些矩陣調整A, B, Pi 後最後回傳一個hmm pointer, 至此算是完成了一個iteration.
在一分鐘內 model 會 不斷的 train, 直到跑了 ”iteration“ 次
或是超過一分鐘時限, 跳出 while loop 執行 dumpHMM.

testing

```c
#include "hmm.h"

int main(int argc, char *argv[]) {

    char *modelList = argv[1];
    char *test_seq = argv[2];
    char *result = argv[3];
    char *compare = "data/test_lbl.txt";
    HMM hmm_initial[5];
    int count[5];
    int max_num = 5;

    start_testing(modelList, &hmm_initial, max_num, test_seq, result);
    double accuracy = calculate_accuracy(compare, result);
    printf("accracy = %f%%\n", accuracy);
    return 0;
}
```

```c
while(1) {
   if(fscanf( seq, "%s", seq_token ) > 0 ) {
      T = strlen(seq_token);
      for(int i = 0 ; i < 5; i++) {
         prob_of_model[i] = viterbi(&hmm[i], seq_token, T);
         // printf("%lf\n", prob_of_model[i]);
         if(max_prob < prob_of_model[i]) {
            max_prob = prob_of_model[i];
            modelNum = i;
            // printf("%d\n", modelNum);
         }
         // printf("%s\n", hmm[modelNum].model_name);
      }
      fprintf(re, "%s", hmm[modelNum].model_name);
      fprintf(re, " %e\n", max_prob);
      modelNum = 0;
      max_prob = -1e9;
   }
   else
      break;
}
```

測試這邊會跑 start_testing (一樣是寫在 hmm.h), 而 calculate_accracy 則是額外用於計算準確率的 function (可以忽略), 整體的準確率也像助教所提供的資料約在80%左右

右圖針對讀進來的每列 sequence 跑一次 viterbi algorithm 找出當前 sequence 在各個 model 中的機率 (看他是屬於哪個 model), 最後輸出結果