メディア情報学実験・音声認識 第二週課題レポート

1510151 柳 裕太

2017 年 11 月 16 日

1 プログラム穴埋め

1.1 forward.c

ソースコード 1: `forward` 関数一部

```
1  /*------------------------------------------
2    初期化
3  ------------------------------------------*/
4  t = 0;
5  for(i=0; i<N; i++) {
6    alpha[t][i] = pi[i] * b[i][O[t]];
7    /* ここを穴埋めした */;
8  }
9
10 /*------------------------------------------
11   再帰計算
12 ------------------------------------------*/
13 for(t=0; t<T-1; t++) {
14   for(j=0; j<N; j++) {
15     sum = 0;
16     for(i=0; i<N; i++) {
17 sum += alpha[t][i] * a[i][j];
18       /* ここを穴埋めした */;
19     }
20     alpha[t+1][j] = sum * b[j][O[t+1]];
21     /* ここを穴埋めした */;
22   }
23 }
24
25 /*------------------------------------------
26   forward確率の計算
```

```
27 ----------------------------------------*/
28 prob = 0.0;
29 for(i=0; i<N; i++) {
30   prob += alpha[t][i];
31   /* ここを穴埋めした */;
32 }
```

1.2 backward.c

ソースコード 2: backward 関数一部

```
1  /*----------------------------------------
2    初期化
3  ----------------------------------------*/
4  t = T-1;
5  for(i=0; i<=N-1; i++) {
6    beta[t][i] = 1
7    /* ここを穴埋めした */;
8  }
9
10 /*----------------------------------------
11   再帰計算
12 ----------------------------------------*/
13 for(t=T-2; t>=0; t--) {
14   for(i=0; i<N; i++) {
15     beta[t][i] = 0;
16     for(j=0; j<N; j++) {
17 beta[t][i] += a[i][j] * b[j][O[t+1]] * beta[t+1][j];
18       /* ここを穴埋めした */;
19     }
20   }
21 }
22
23 /*----------------------------------------
24   backward確率の計算
25 ----------------------------------------*/
26 prob = 0.0;
27 for(i=0; i<N; i++) {
28   prob += pi[i] * b[i][O[1]] * beta[1][i];
29   /* ここを穴埋めした */;
30 }
```

```
31 */
```

1.3 baumwelch.c

ソースコード 3: baumWelch 関数一部

```
1         /*--------------------------------------
2           π i を再推定
3         --------------------------------------*/
4         for(i=0; i<N; i++)
5           pi[i] = (alpha[1][i] * beta[1][i]) / prob_old;
6           /*ここを穴埋め(Algorithm 1の第17行)*/;
7
8
9         /*--------------------------------------
10          A (a_ij) を再推定
11        --------------------------------------*/
12        for(i=0; i<N; i++) {
13          for(j=0; j<N; j++) {
14        sum1 = sum2 = 0.0;
15            for(t=1; t<=T-1; t++){
16              sum1 += alpha[t][i] * a[i][j] * b[j][O[t+1]] * beta[t
                    +1][j];
17              sum2 += alpha[t][i] * beta[t][j];
18            }
19            a[i][j] = sum1 / sum2;
20           /*
21              ここを穴埋めした
22              (Algorithm 1の第21行)*
23            */
24          }
```

2 各プログラム実行結果

2.1 forward

ソースコード 4: forward 実行結果

```
1   [~/asr/drill]\% ./forward
2   0.0000000 0.0000000 0.0182400 0.0634368
3   0.0000000 0.0320000 0.0825600 0.0588096
```

```
0.0000000 0.3840000 0.1459200 0.0120192
0.8000000 0.0480000 0.0028800 0.0006912
P(O|λ)= 0.1349568
```

2.2 backward

ソースコード 5: backward 実行結果

```
[~/asr/drill]\% ./backward
0.0630000 0.2100000 0.7000000 1.0000000
0.0713880 0.2274000 0.6700000 1.0000000
0.1474600 0.2990000 0.4500000 1.0000000
0.1686960 0.2680000 0.4200000 1.0000000
P(O|λ)= 0.0536000
```

2.3 baumwelch.c

ソースコード 6: baumwelch 実行結果

```
[~/asr/drill]\% baumwelch < d1.txt
Before Training ----------
pi= 1.000000 0.000000
A=
0.500000 0.500000
0.500000 0.500000
B=
0.500000 0.500000
0.500000 0.500000

Baum-Welch estimation converged.
After Training ----------
pi= 1.000000 0.000000
A=
0.256001 0.752594
0.769426 0.251694
B=
0.805410 0.194590
0.232469 0.767531


[~/asr/drill]\% baumwelch < d2.txt
```

```
Before Training ----------
pi= 1.000000 0.000000
A=
0.500000 0.500000
0.500000 0.500000
B=
0.500000 0.500000
0.500000 0.500000

Baum–Welch estimation converged.
After Training ----------
pi= 1.000000 0.000000
A=
0.730665 0.278419
0.274128 0.734279
B=
0.828889 0.171111
0.200598 0.799402
```

2.4 recogd

ソースコード 7: recogd 実行結果

```
[~/asr/drill]\% recogd data1.list > data1.result
n1= 93
n2= 7
[~/asr/drill]\% recogd data2.list > data2.result
n1= 7
n2= 93
```

また、recogd の実行結果をグラフ化させたのは以下の図 1 の通り。

3 実験のポイント

今回の実験では、隠れマルコフモデル (HMM) の $P(O|\ \lambda)$ の効率的な算法である Forward アルゴリズムと Backward アルゴリズムを、手元の手計算と C 言語アルゴリズム化の 2 方向から体験した。また、それを元に実際にパラメータ推定 (Baum-Welch) アルゴリズムの C 言語上での実装を行うことで、実際に HMM による認識実験を行った。

最大のポイントは、Forward・Backward アルゴリズムの内容を深く理解する点にあると考えている。これは、効率的な $P(O|\ \lambda)$ の導出方法を知ることで、演算回数を大幅に減らすことで、一気に機械認識のハードルを下げることにつながるためである。

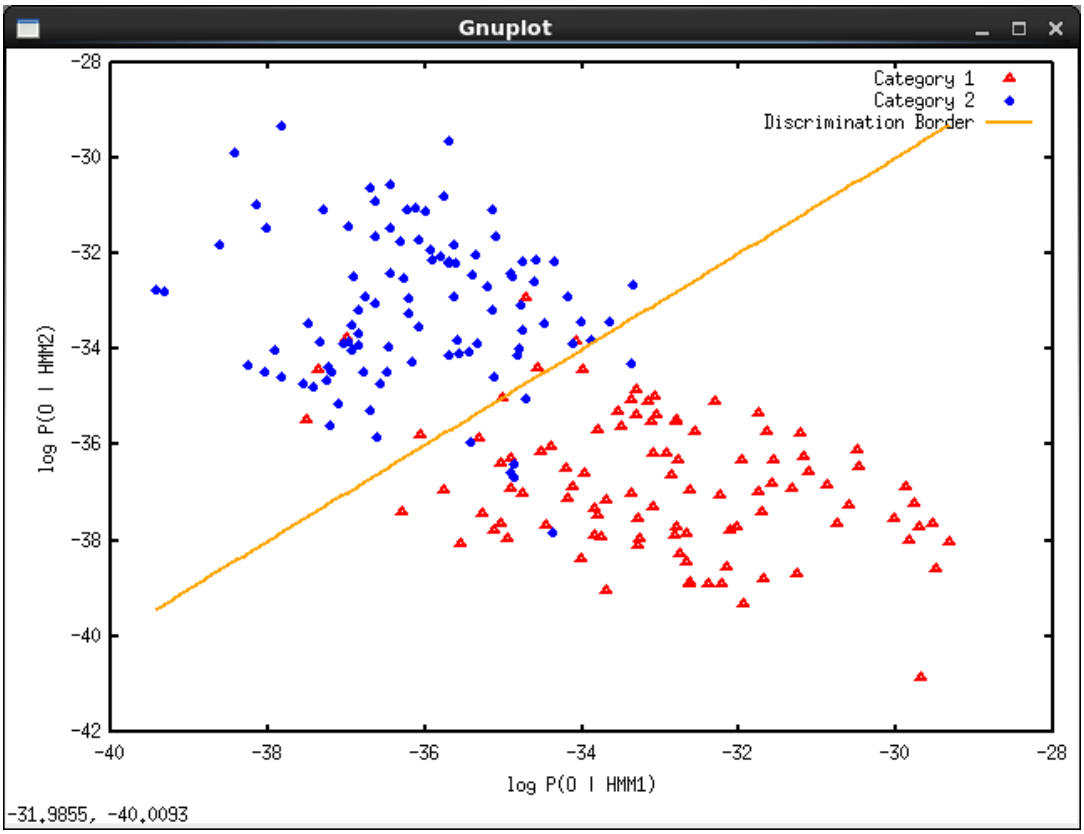

図 1: 実行結果グラフ

4 よくわかったこと

Forward・Backward アルゴリズムの挙動を、手計算・C 言語実装で深く理解することができた。

5 よくわからなかったこと

Forward・Backward アルゴリズムによって $P(O|\ \lambda)$ の導出への負担を大幅に減らすことができる…らしいが、実際どれぐらい処理速度が速くなるのか知ることができなかった。

6 要望

baumwelch・recogd プログラム実装には、想定より多くの落とし穴があった (値変更忘れ、for 文の繰り返し変数上限・下限の設定など)。この影響でミスに気付くのに時間がとられ、作業効率が落ちてしまった。これらに関しては TA チェックの際に、チェックリストのようなものがあると、迅速にミスに気付ける可能性がある。

7 感想・その他

来週、HMM を活用した音声認識を行うということで、どのようなものになるか楽しみである。