

語音處理 HW1 Hidden Markov Models

Homework 1

- A three-state Hidden Markov Model for the *Dow Jones Industrial average*

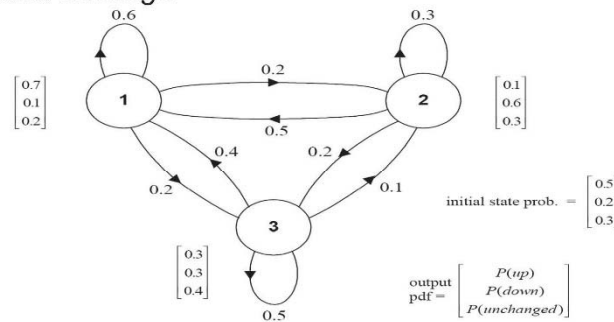


Figure 8.2 A hidden Markov model for the Dow Jones Industrial average. The three states no longer have deterministic meanings as the Markov chain illustrated in Figure 8.1.

- Find the probability:
 $P(\text{up, up, unchanged, down, unchanged, down, up}|\lambda)$
- Find the optimal state sequence of the model which generates the observation sequence: (up, up, unchanged, down, unchanged, down, up)

SP - Berlin Chen 42

-Find the probability $P(\text{up, up, unchanged, down, unchanged, down, up}|\lambda)$

Question1:

$$\begin{aligned}
 &P(\text{up, up, unchanged, down, unchanged, down, up}|\lambda) \\
 &= P(S1)P(S1|S1)P(S3|S1)P(S2|S3)P(S3|S2)P(S2|S3)P(S1|S2) \\
 &= 0.5 \ 0.6 \ 0.2 \ 0.1 \ 0.2 \ 0.1 \ 0.5 \\
 &= 0.00006
 \end{aligned}$$

-Find the optimal state sequence of the model which generates the observation sequence: (up, up, unchanged, down, unchanged, down, up)

Question2:

$$\begin{aligned}
 &\text{Best case of observation sequence: (up, up, unchanged, down, unchanged, down, up)} \\
 &= \text{State sequence ['s1', 's1', 's3', 's3', 's3', 's3', 's1']} \\
 &= P(S1, \text{up})P(S1, \text{up} | S1)P(S3, \text{unchanged} | S1)P(S3, \text{down} | S3)P(S3, \text{unchanged} | S3)P(S3, \text{down} | S3)P(S1, \text{up} | S3) \\
 &= 0.7 \ 0.5 \ 0.6 \ 0.7 \ 0.2 \ 0.4 \ 0.5 \ 0.3 \ 0.5 \ 0.4 \ 0.5 \ 0.3 \ 0.4 \ 0.7 \\
 &= 0.0000148176
 \end{aligned}$$

B: 程式運算結果

使用 javascript 進行撰寫，運行結果如下

```

C:\Windows\system32\cmd.exe

[ 's1', 's1', 's3', 's3', 's3', 's3', 's1' ]

C:\Users\Mike\git\Course-SpeechProcessing\HW1>node hw1-hmm.js

Init_state prob:
{ s1: 0.5, s2: 0.2, s3: 0.3 }

Transition prob:
{ s1: { s1: 0.6, s2: 0.2, s3: 0.2 },
  s2: { s1: 0.5, s2: 0.3, s3: 0.2 },
  s3: { s1: 0.4, s2: 0.1, s3: 0.5 } }

Observations prob:
{ s1: { up: 0.7, down: 0.1, unchanged: 0.2 },
  s2: { up: 0.1, down: 0.6, unchanged: 0.3 },
  s3: { up: 0.3, down: 0.3, unchanged: 0.4 } }

Question1: P(up, up, unchanged, down, unchanged, down, up | λ)
          = P(S1)P(S1|S1)P(S3|S1)P(S2|S3)P(S3|S2)P(S2|S3)P(S1|S2)
          = 0.5 0.6 0.2 0.1 0.2 0.1 0.5
          = 0.00006

Question2: Find the optimal state sequence of the model which generates the ob
servations sequence: (up, up, unchanged, down, unchanged, down, up)

BruteForce:
          = 0.0000148176
          [ 's1', 's1', 's3', 's3', 's3', 's3', 's1' ]

Greedy:
          = 0.00000889056
          [ 's1', 's1', 's1', 's2', 's1', 's2', 's1' ]

Viterbi:
          = 0.0000148176
          [ 's1', 's1', 's3', 's3', 's3', 's3', 's1' ]

C:\Users\Mike\git\Course-SpeechProcessing\HW1>_

```

C: 程式碼

細節步驟及詳細撰寫紀錄已放上 git

<https://github.com/Mike-Zheng/Course-SpeechProcessing/tree/master/HW1>

隱馬可夫數學模型:

```
//2016 NTU mike
//隱馬可夫數學模型
var HMM = {
  data: {
    //觀測符號物件
    //observations
    obs: [],
    //狀態物件
    //states
    states: [
      's1',
      's2',
      's3'
    ],
    //初始機率物件
    //init_state_prob
    init_prob: {
      's1': 0.5,
      's2': 0.2,
      's3': 0.3
    },
    //狀態轉移機率物件
    //transition_prob
    trans_prob: {
      's1': { 's1': 0.6, 's2': 0.2, 's3': 0.2 },
      's2': { 's1': 0.5, 's2': 0.3, 's3': 0.2 },
      's3': { 's1': 0.4, 's2': 0.1, 's3': 0.5 }
    },
    //觀測符號出現機率
    //observations_prob
    obs_prob: {
      's1': { 'up': 0.7, 'down': 0.1, 'unchanged': 0.2 },
      's2': { 'up': 0.1, 'down': 0.6, 'unchanged': 0.3 },
      's3': { 'up': 0.3, 'down': 0.3, 'unchanged': 0.4 }
    }
  },
  setObservations: function(arr) {
    //以物件導向方式設定 observations 值
    this.data.obs = arr;
  },
  printInit: function() {
    //將模型內數學機率印出
    console.log("\n Init_state prob: \n", this.data.init_prob);
    console.log("\n Transition prob: \n", this.data.trans_prob);
    console.log("\n Observations prob: \n", this.data.obs_prob);
  },
  //第一題
  question1: function() {
```

```

    var initP = this.data.init_prob;
    var tranP = this.data.trans_prob;
    //乘法佇列
    var sequence = [initP['s1'], tranP['s1']['s1'], tranP['s1']['s3'], tranP['s3']['s2'],
    tranP['s2']['s3'], tranP['s3']['s2'], tranP['s2']['s1']];
    console.log("\nQuestion1:  P(up, up, unchanged, down, unchanged, down, up|\lambda)");
    console.log("    =P(S1)P(S1|S1)P(S3|S1)P(S2|S3)P(S3|S2)P(S2|S3)P(S1|S2)");
    console.log("    =", sequence.join(" "));
    var ans = 1.0;
    //浮點數相乘
    sequence.forEach(function(p, i) {
        ans = accMul(ans, p);
    });
    console.log("    =", ans);

},
//第二題
question2: function(prob) {
    this.setObservations(prob);
    var q2 = "\nQuestion2:  Find the optimal state sequence of the model which generates the
observation sequence: (up, up, unchanged, down, unchanged, down, up)";
    console.log(q2);
    BruteForce(this.data);
    greedy(this.data);
    Viterbi_a(this.data);
}
}

```

執行:

```

HMM.printInit();
HMM.question1();
HMM.question2(['up', 'up', 'unchanged', 'down', 'unchanged', 'down', 'up']);

```

BruteForce 演算法(暴力解):

```

function BruteForce(data) {

    //排列組合用的維度 BEGIN
    //參考 http://blog.darkthread.net/post-2012-03-17-recursion-game.aspx
    var dimensions = [];
    for (var i = 0; i < 7; i++) {
        dimensions.push(["s1", "s2", "s3"]);
    }
    //用以存放結果的陣列
    var results = [];
    //使用遞迴方式排列出所有組合
    function explore(curDim, prefix) {
        //取出下一層維度
        var nextDim = dimensions.shift();
        for (var i = 0; i < curDim.length; i++) {
            if (nextDim)
                //若仍有下一層維度，繼續展開

```

```

        explore(nextDim, prefix + curDim[i] + ",");
    else
        //若已無下一層，則傳入字首加上目前維度選項成為結果
        results.push(prefix + curDim[i]);
    }
    //將下層維度存回，供上層維度其他選項使用
    if (nextDim) dimensions.push(nextDim);
}
//傳入第一層維度開始演算
explore(dimensions.shift(), "");
//排列組合用的維度 END

//初始化最大值暫存變數
var MAX = 0;
//初始化佇列
var sq;
var nowAns = 1;

//從 1~3^7 將所有路徑納入考慮
for (var i = 0; i < 2187; i++) {
    var dig = results[i].split(',');
    nowAns = 1;
    for (var j = 0; j < 7; j++) {
        //將路徑上機率相乘
        nowAns = accMul(data.obs_prob[dig[j]][data.obs[j]], nowAns);
        //乘上一開始初始機率
        if (j == 0) nowAns = accMul(data.init_prob[dig[j]], nowAns);
        if (j < 6) nowAns = accMul(data.trans_prob[dig[j]][dig[j + 1]], nowAns);
    }
    if (MAX < nowAns) {
        //找出可能機率放入暫存變數
        MAX = nowAns;
        sq = results[i];
    }
}

console.log("\nBruteForce: ");
console.log("    =", MAX);
console.log("    ", sq.split(',').join().split(','));

//檢查答案局部乘法用
// console.log('Check: ');
// var qdi = sq.split(',')
// // console.log(data.obs_prob[qdi[0]]);
// nowAns = 1;
// for (var j = 0; j < 7; j++) {
//     console.log(data.obs_prob[qdi[j]][data.obs[j]]);
//     if (j == 0) console.log(data.init_prob[qdi[j]]);
//     if (j < 6) console.log(data.trans_prob[qdi[j]][qdi[j + 1]]);
// }
}

```

Greedy 演算法:

```

function greedy(data) {
  var sq = [];
  var nowAns = 1;
  var step = data.obs;

  //加入路徑
  sq.push(syToSt(step[0]));
  // 初始狀態出現機率
  nowAns = accMul(data.obs_prob[syToSt(step[0])][step[0]], data.init_prob[syToSt(step[0])]);

  for (var i = 1; i < step.length; i++) {
    //將 7 個節點抓該點最大的相乘
    nowAns = accMul(findMAX(data.trans_prob, sq[sq.length - 1], step[i]), nowAns);
  }

  //symbol 轉 state
  function syToSt(inp) {
    var state;
    if (inp == 'up' || inp == 0) state = 's1';
    else if (inp == 'down' || inp == 1) state = 's2';
    else if (inp == 'unchanged' || inp == 2) state = 's3';
    return state;
  }

  //找這個節點最大的
  function findMAX(arr, now, nxtObs) {
    var stateTemp;
    var max = 0;
    var fd;

    for (var i = 0; i < data.states.length; i++) { //state 有 3 種
      //考慮結點挑最大的
      fd = syToSt(i);
      if (accMul(data.trans_prob[now][fd], data.obs_prob[fd][nxtObs]) > max) {
        max = accMul(data.trans_prob[now][fd], data.obs_prob[fd][nxtObs]);
        stateTemp = fd;
      }
    }
    //若為最大加入路徑
    sq.push(stateTemp);
    return max;
  }

  console.log("\nGreedy: ");
  console.log("    =", nowAns);
  console.log("    ", sq);
}

```

Viterbi 演算法:

```

function Viterbi_a(data) {
  var path = [];
  var step = data.obs;
  var sq = []; //答案路徑
  var Ans; //最佳答案
  //[{s1:w,s2:w,s3:w},{}]
  //初始化路徑
  path[0] = {
    's1': accMul(data.obs_prob['s1'][step[0]], 0.5),
    's2': accMul(data.obs_prob['s2'][step[0]], 0.2),
    's3': accMul(data.obs_prob['s3'][step[0]], 0.3)
  };
  for (var i = 1; i < step.length; i++) {
    //findMAX(i,'s1')
    //找出最大的路徑權重
    path[i] = {
      's1': accMul(data.obs_prob['s1'][step[i]], findMAX(i, 's1')),
      's2': accMul(data.obs_prob['s2'][step[i]], findMAX(i, 's2')),
      's3': accMul(data.obs_prob['s3'][step[i]], findMAX(i, 's3'))
    };
  }

  //權重步驟
  // console.log(path);

  //最後算出來最大的權重就是答案
  Ans = max(path[step.length - 1]['s1'], path[step.length - 1]['s2'], path[step.length - 1]['s3'])[0];
  //向前找路徑
  //初始化最後路徑
  var tempAns = Ans;
  var tempSt = max(path[step.length - 1]['s1'], path[step.length - 1]['s2'], path[step.length - 1]['s3'])[1];
  sq.unshift(max(path[step.length - 1]['s1'], path[step.length - 1]['s2'], path[step.length - 1]['s3'])[1]);

  //往回找路
  for (var i = 5; i >= 0; i--) {
    // accMul(path[i]['s1'], data.trans_prob['s1'][nowSt])
    for (var k = 0; k < 3; k++) {
      // console.log(accMul(path[i][syToSt(k)], data.trans_prob[syToSt(k)][tempSt]));
      if (accMul(data.obs_prob[tempSt][step[i + 1]], accMul(path[i][syToSt(k)], data.trans_prob[syToSt(k)][tempSt])) == tempAns) {
        tempAns = path[i][syToSt(k)];
        tempSt = syToSt(k);
        sq.unshift(syToSt(k));
      }
    }
  }

  }

  console.log("\nViterbi: ");
  console.log("  =", Ans);
  console.log("  ", sq);

```

```
function max(a, b, c) {
    var M = Math.max(a, b, c);
    if (M == a) return [M, 's1'];
    else if (M == b) return [M, 's2'];
    else if (M == c) return [M, 's3'];
}

//symbol 轉 state
function syToSt(inp) {
    var state;
    if (inp == 'up' || inp == 0) state = 's1';
    else if (inp == 'down' || inp == 1) state = 's2';
    else if (inp == 'unchanged' || inp == 2) state = 's3';
    return state;
}

//找這個節點最大的
function findMAX(i, nowSt) {
    // var stateTemp;
    var max = 0;
    // 找出(前一節點)X(轉移機率)權重最大的
    for (var j = 0; j < 3; j++) {
        if (accMul(path[i - 1][syToSt(j)], data.trans_prob[syToSt(j)][nowSt]) > max)
            max = accMul(path[i - 1][syToSt(j)], data.trans_prob[syToSt(j)][nowSt]);
    }
    return max;
}
}
```


D: 時間複雜度計算

這次演算法的撰寫，主要針對三種演算法進行攻略

演算法	攻略細節	預計時間複雜度
BruteForce 暴力解	暴力解，將 3^7 路徑都進行進算	$O(n^m)$
Greedy 貪婪演算法	透過找尋下一個最大的可能進行計算，算出來的結果是錯的，但卻很快可以找到一個可能的答案。	$O(n^2)$
Viterbi dynamic programming	Viterbi 透過路徑權重記錄進行運算，算是用空間取得時間的一種方式，但由於從中無法紀錄運算的步驟(太浪費空間)，所以最後還要從解果逆推回去計算進行的步驟。	$O(2n^2)$

E: 參考

陳柏琳教授課程網站

http://berlin.csie.ntnu.edu.tw/Courses/Speech%20Processing/Speech%20Processing_Main_2016S.htm

生物信息学课件 Hidden Markov Model 隐马尔可夫模型

<http://www.tudou.com/programs/view/cBZqDKojiw8/>

HMM 学习笔记_1(从一个实例中学习 DTW 算法)

<http://www.cnblogs.com/tornadomeet/archive/2012/03/23/2413363.html>

演算法筆記- Hidden Markov Model

<http://www.csie.ntnu.edu.tw/~u91029/HiddenMarkovModel.html>

F: 心得

這次撰寫過程著實花了一些時間，從理解 HMM 到攻略演算法，都花了不少時間推導。

主要撰寫的 pattern 是將各個機率做成一個數學模型，再透過不同的演算法 function，進行運算。

基於目前的實習工作及預計想走的方向，使用 javascript(nodejs)目前前後端都流行的語言進行撰寫開

發，其中浮點數相乘遇到的問題解法，及暴力解的狀態排列組合序列(遞回部分)，有參考網路上的解法，

剩下部分都是自行完成，程式碼的撰寫有上 git，如需運行需要安裝 nodeJs 做為直譯工具。

<https://github.com/Mike-Zheng/Course-SpeechProcessing/tree/master/HW1>