HMM レポート

平成 28 年 7 月 24 日
未来ロボティクス学科
B3 1426015
今井 良佑

1 目的

今回のレポートでは隠れマルコフモデル（以降 HMM）の認識アルゴリズムや学習アルゴリズムについて理解し認識についてツボとボールを用いたシュミレーションを作成する．

2 理論[1]

2.1 マルコフモデルと隠れマルコフモデル

まず，マルコフモデルとは観測値が過去のデータに影響して決定されるものを指す．
マルコフモデルで x_n の確率は一般的に以下の式 (1) ように表される．
ここで x_nは

$$x_n = \begin{pmatrix} x_{n1} \\ x_{n2} \\ \vdots \\ x_{nK} \end{pmatrix}$$

$$x_{nk} \in \{0, 1\}, \forall n \in \{1, \cdots, N\}, \forall k \in \{1, \cdots, K\}$$

$$\sum_{k=1}^{K} x_{nk} = 1, \forall n \in \{1, \cdots, N\}$$

をなる K 次元のベクトルである．K というのは，出力の種類である．

$$p(x_n) = p(x_n|x_1, \cdots, x_{n-1}) \tag{1}$$

これはつまり x_n にはそれまでの x_1から x_{n-1} までの出力も重要であるということである．x_n が一つ前の値のみに依存しそれ以外の事象からは独立している場合は，

$$p(x_n) = p(x_n|x_{n-1}) \tag{2}$$

となる．これを一次マルコフ連鎖と呼ぶ．$x_1, \cdots, x_N$ の系列が出力される確率は，

$$p(x_1, \cdots, x_N) = p(x_1) \prod_{n=2}^{N} p(x_n|x_{n-1}) \tag{3}$$

となる．隠れマルコフモデルとは出力 x_n に対してそれに対応する潜在変数 z_n がある．z_nは

$$z_n = \begin{pmatrix} z_{n1} \\ z_{n2} \\ \vdots \\ z_{nK} \end{pmatrix}$$

$$z_{nk} \in \{0, 1\}, \forall n \in \{1, \cdots, N\}, \forall k \in \{1, \cdots, K\}$$

$$\sum_{k=1}^{K} z_{nk} = 1, \forall n \in \{1, \cdots, N\}$$

をなる K 次元のベクトルである．

x_n は z_n により影響を受ける，また z_n は一次マルコフ連鎖となっている．これを式で表すと以下の式 (4) のようになる．

$$p(x_1, \cdots, x_N, z_1, \cdots, z_N) = p(z_1)\left[\prod_{n=2}^{N} p(z_n|z_{n-1})\right]\prod_{n=1}^{N} p(x_n|z_n) \tag{4}$$

ここで式 (4) の右辺について説明をすると，$p(z_1)$ は潜在変数の初期位置が z_1 である確率，$p(z_n|z_{n-1})$ は潜在変数 z_{n-1}から z_n という一次マルコフ連鎖となる確率，$p(x_n|z_n)$ は潜在変数 z_nから出力 x_n が出力される確率である．

ここでいくつか変数を導入する．

状態推移確率をまとめた $\mathrm{A} = \mathbb{R}^{K\times K}$,

$$p(z_n|z_{n-1}, A) = \prod_{k=1}^{K}\prod_{j=1}^{K} A_{jk}^{z_{n-1,j}z_{nk}} \tag{5}$$

初期状態確率 $\pi = \mathbb{R}^{1\times K}$,

$$p(z_1|\pi) = \prod_{k=1}^{K} \pi_k^{z_{1k}} \tag{6}$$

それぞれの状態に対応した出力の確率を示した ϕ を使用する．

また，ϕ については設定するモデルが離散か連続かなどにより異なる．

これらの変数をまとめて $\theta = \{\pi, A, \phi\}$ とする

以上より式 (4) を書き換えると式 (7) になる．

$$p(X, Z|\theta) = p(z_1|\pi)\left[\prod_{n=2}^{N} p(z_n|z_{n-1}|A)\right]\prod_{m=1}^{N} p(x_m|z_m, \phi) \tag{7}$$

2.2 forward アルゴリズムと vitabi アルゴリズム

次に適当な HMM モデルが存在するときその実現確率を考える．なお簡単のため状態の遷移の順番が決まっていて戻らない left-to-right HMM とする．

しかし出力された系列の潜在変数がどの状態なのかは出力を見るだけではわからない．そのため例えば，状態 3 のモデルで出力が 8 つ確認されたとするこの場合考えられるパターンは ${}_7\mathrm{C}_2 = 21$ 通りある．もしモデルの数が i 出力数が j ならばパターンは ${}_{j-1}\mathrm{C}_{i-1}$ 通りとなる．これらをすべて列挙するのは非効率的である．そこで図 1 に示す横軸に出力，縦軸に状態をとったトレリスというものを使う．

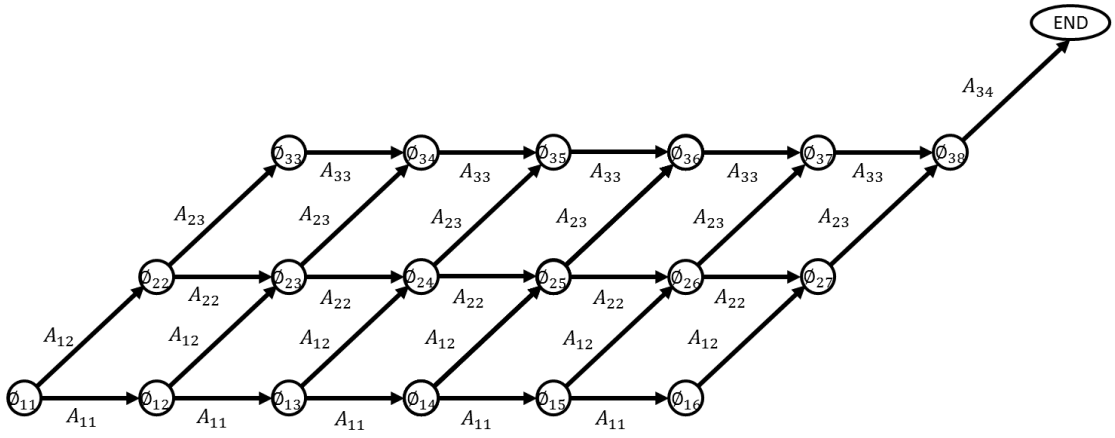

図 1: トレリス図

状態 i，出力 j 番目の時までの出現確率 $\alpha_{(i,j)}$ を求める．この時に forward アルゴリズム (式 (8)) と vitabi アルゴリズム (式 (9)) の二種類の方式がある．

$$\alpha_{(i,j)} = \sum_{k=0}^{1} \alpha_{(i-k,j-1)} A_{i-k,i} \phi_{i,j} \tag{8}$$

$$\alpha_{(i,j)} = max(\alpha_{(i-k,j-1)} A_{i-k,i} \phi_{i,j}) \quad k \in \{0,1\} \tag{9}$$

これを繰り返すことにより出現確率が求まる．

2.3 EM アルゴリズム

2.3.1 導出

HMM の変数 θ を適切なものにするために用いられるのが EM アルゴリズムである．
まずパラメータの初期値を θ^{old} とする．
尤度関数の対数の期待値を $Q(\theta, \theta^{old})$ で定義する．

$$Q(\theta, \theta^{old}) = \sum_{Z} p(Z|X, \theta^{old}) \ln p(X, Z|\theta) \tag{10}$$

EM アルゴリズムには E ステップと M ステップがある．
E ステップは後述する $\gamma(z_n)$ と$\xi(z_{n-1}, z_n)$ をもとめるステップ
M ステップは $\gamma(z_n)$ と$\xi(z_{n-1}, z_n)$ を定数とみなし，$\theta = \{\pi, A, \phi\}$ に対して $Q(\theta, \theta^{old})$ を最大化するステップである．

$\gamma(z_n)$ を潜在変数 z_n の周辺事後分布 $\xi(z_{n-1}, z_n)$ を 2 つの連続した潜在変数に対する同時事後分布とする．

$$\gamma(z_n) = p(z_n|X, \theta^{old}) \tag{11}$$

$$\xi(z_{n-1}, z_n) = p(z_{n-1}, z_n|X, \theta^{old}) \tag{12}$$

また，$\gamma(z_{nk})$,$\xi(z_{n-1,j} z_{nk})$ を以下のように定義する．

$$\gamma(z_{nk}) = \mathbb{E}[z_{nk}] = \sum_{z_n} \gamma(z_n) z_{nk} \tag{13}$$

$$\xi(z_{n-1,j} z_{nk}) = \mathbb{E}[z_{n-1}, z_n] = \sum_{z_{n-1}, z_n} \xi(z_{n-1}, z_n) z_{n-1,j} z_{nk} \tag{14}$$

ここでまず式 (7) を式 (10) に代入し，γ,ξ の定義を用い変形すると以下の式 (15) を得る．

$$Q(\theta, \theta^{old}) = \sum_{k=1}^{K} \gamma(z_{1k}) \ln \pi_k + \sum_{n=2}^{N} \sum_{j=1}^{K} \sum_{k=1}^{K} \xi(z_{n-1,j}, z_{nk}) \ln A_{jk} + \sum_{n=2}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \ln p(x_n|\phi_k) \tag{15}$$

2.3.2 M ステップ

A と π には拘束条件

$$\sum_{k=1}^{K} A_{jk} = 1, \forall j \in \{1, \cdots, K\}$$

$$\sum_{k=1}^{K} \pi_k = 1$$

が存在する．
ここからラグランジュの未定乗数 $(\lambda_1, \cdots, \lambda K + 1)$ を導入し拘束条件付きの最大化問題として $Q(\theta, \theta^{old})$ の最大化を行う．

$$\theta_{max} = argmax_{\theta=(\pi,A,\phi)} F(\pi, A, \phi, \lambda) \tag{16}$$

$$F(\pi, A, \phi, \lambda) = Q(\theta, \theta^{old}) - \lambda_1 \left(\sum_{k=1}^{K} \pi_k - 1 \right) - \sum_{j=1}^{K} \lambda_{j+1} \left(\sum_{k=1}^{K} A_{jk} - 1 \right) \tag{17}$$

$F(\pi, A, \phi, \lambda)$ をπ，A で偏微分することで

$$\frac{\partial F}{\partial \pi_k} = \frac{\gamma(z_{1k})}{\pi_k} - \lambda_1 \tag{18}$$

$$\frac{\partial F}{\partial A_{jk}} = \sum_{n=2}^{N} \frac{\xi(z_{n-1,j}, z_{nk})}{A_{jk}} - \lambda_{j+1} \tag{19}$$

これらの偏微分の値が 0 になるようにすればいいので，π，A について解くと，

$$\pi_k = \frac{\gamma(z_{1k})}{\pi_k} \tag{20}$$

$$A_{jk} = \sum_{n=2}^{N} \frac{\xi(z_{n-1,j}, z_{nk})}{\lambda_{j+1}} \tag{21}$$

式 (20)(21) を拘束条件の式に代入すると，

$$\frac{\gamma(z_{1k})}{\pi_k} = 1$$

$$\lambda_1 = \sum_{k=1}^{K} \gamma(z_{1k}) \tag{22}$$

$$\sum_{k=1}^{K}\sum_{n=2}^{N} \frac{\xi(z_{n-1,j}, z_{nk})}{\lambda_{j+1}} = 1$$

$$\lambda_{j+1} = \sum_{k=1}^{K}\sum_{n=2}^{N} \xi(z_{n-1,j}, z_{nk}) \tag{23}$$

式 (23)(24) を式 (20)(21) に代入して，

$$\pi_k = \frac{\gamma(z_{1k})}{\pi_k} \tag{24}$$

$$A_{jk} = \sum_{n=2}^{N} \frac{\xi(z_{n-1,j}, z_{nk})}{\sum_{k=1}^{K}\sum_{n=2}^{N} \xi(z_{n-1,j}, z_{nk})} \tag{25}$$

以上が変数の更新式となる．残るは $\gamma(z_{1k})$ と$\xi(z_{n-1,j}, z_{nk})$ の効率的な算出方法がわかれば学習ができる．ここからは参考文献に倣い $p(X, z_n) = p(X, z_n|\theta)$ という略記を用いる．

$$p(X, z_n) = p(x_1, \cdots, p_{x_n}|z_n)p(x_{n+1}, \cdots, x_N|z_n)p(z_n)$$

$$\alpha(z_n) = p(x_1, \cdots, x_n, z_n) \tag{26}$$

$$\beta(z_n) = p(x_{n+1}, \cdots, x_N|z_n) \tag{27}$$

を用いると

$$\begin{aligned}\gamma(z_n) &= p(z_n|X) \\ &= \frac{p(X, z_n)}{p(X)} \\ &= \frac{p(x_1, \cdots, p_{x_n}|z_n)p(x_{n+1}, \cdots, x_N|z_n)p(z_n)}{p(X)} \\ &= \frac{\alpha(z_n)\beta(z_n)}{p(X)}\end{aligned} \tag{28}$$

$\alpha(z_n)$ を式変形し効率的に求めると以下のようになる．

$$\begin{aligned}\alpha(z_n) &= p(x_1, \cdots, x_n, z_n) \\ &= p(x_n|z_n) \sum_{z_{n-1}} p(x_1, \cdots, x_{n-1}, z_{n-1}) p(z_n|z_{n-1}) \end{aligned} \tag{29}$$

式 (26) より

$$\alpha(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \alpha(z_{n-1}) p(z_n|z_{n-1}) \tag{30}$$

のように再帰的に解ける．また

$$\alpha(z_1) = p(x_1|z_1) p(z_1) \tag{31}$$

である．
同様に $\beta(z_n)$ についても

$$\begin{aligned}\beta(z_n) &= p(x_{n+1}, \cdots, x_N|z_n) \\ &= \sum_{z_{n+1}} p(x_{n+2}, \cdots, x_N|z_n) p(x_{n+1}|z_{n+1}) p(z_{z+1}|z_n) \end{aligned} \tag{32}$$

式 (27) より

$$\beta(z_n) = \sum_{z_{n+1}} \beta(z_{n+1}) p(x_{n+1}|z_{n+1}) p(z_{z+1}|z_n) \tag{33}$$

また，$\gamma(z_n) = \frac{\alpha(z_N)\beta(z_N)}{p(X)}$ と $p(z_n|x) = \frac{p(X,z_N)\beta(z_N)}{p(X)}$ より，

$$\beta(z_1) = 1 \tag{34}$$

これらから $\gamma(z_n)$,$\xi(z_{n-1}, z_n)$ の更新式を求めると，

$$\gamma(z_n) = \frac{\alpha(z_N)\beta(z_N)}{p(X)} \tag{35}$$

$$\xi(z_{n-1}, z_n) = \frac{\alpha(z_{n-1})\beta(z_n) p(x_n|z_n) p(z_n|z_{n-1})}{p(X)} \tag{36}$$

この式により $\gamma(z_n)$,$\xi(z_{n-1}, z_n)$ を求める．

2.3.3 要約

(1) $\theta^{old} = \pi, A, \phi$ の初期値を定める

(2)

$$\begin{cases} \alpha(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \alpha(z_{n-1}) p(z_n|z_{n-1}) \\ \alpha(z_1) = p(x_1|z_1) p(z_1) \end{cases}$$

$$\begin{cases} \beta(z_n) = \sum_{z_{n+1}} \beta(z_{n+1}) p(x_{n+1}|z_{n+1}) p(z_{z+1}|z_n) \\ \beta(z_1) = 1 \end{cases}$$

から，

$$\begin{cases} \gamma(z_n) = \frac{\alpha(z_N)\beta(z_N)}{p(X)} \\ \xi(z_{n-1}, z_n) = \frac{\alpha(z_{n-1})\beta(z_n) p(x_n|z_n) p(z_n|z_{n-1})}{p(X)} \end{cases}$$

を求める．

(3)

$$\pi_k = \frac{\gamma(z_{1k})}{\pi_k} \tag{37}$$

$$A_{jk} = \sum_{n=2}^{N} \frac{\xi(z_{n-1,j}, z_{nk})}{\sum_{k=1}^{K}\sum_{n=2}^{N}\xi(z_{n-1,j}, z_{nk})} \tag{38}$$

でパラメータの更新を行う

3 環境・使用機器

今回 HMM の認識シュミレーションを行うにあたって Python で書いた以下のソースコード 1 を使用する.

ソースコード 1: HMM.py

```python
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------------
# Name:  MFCC.py
# Author:  R.Imai
# Created: 2016 / 06 / 22
# Last Date: 2016 / 07 / 24
# Note:
#-------------------------------------------------------------------------------
import random
import numpy as np

class HMM:
    def __init__(self, transProb, stayProb, valProb, label):
        self.transProb = transProb #次に行く確率
        self.stayProb = stayProb #居座る確率
        self.label = label
        self.valProbList = valProb
        self.setValProb(valProb) #出力確率
        self.state = 0
        self.stateNum = len(transProb)
        self.valNum = len(valProb)

    def showState(self):
        for i in range(self.stateNum):
            print("----------pot" + str(i) + "----------")
            print("\t----------")
            for label in self.label:
                print("\t" + label + ": " + str(self.valProb[i][label]))
            print("\t----------")
            print("trans probability" + str(self.transProb[i]))
            print("stay probability" + str(self.stayProb[i]))
            print("------------------------")

    def setValProb(self, valprob):
        self.valProb = [{} for i in range(len(valprob))]
        for j in range(len(valprob)):
            for i in range(len(self.label)):
                self.valProb[j][self.label[i]] = valprob[j][i]


    def forward(self, inputModel):
        colLen = self.stateNum
        rowLen = len(inputModel) - colLen + 1
        workspace = [[0 for i in range(colLen)] for j in range(rowLen)]
        #print(str(len(workspace)) + ", " + str(len(workspace[0])))
        #print(workspace[colLen - 1][rowLen - 1])
        calcSpace = [0, 0]
        workspace[0][0] = self.valProb[0][inputModel[0]]
        for i in range(1, rowLen):
            #print(str(i) + "/" + str(rowLen))
            workspace[i][0] = workspace[i - 1][0]*self.stayProb[0]*self.valProb[0][inputModel[i]]
        for j in range(1, colLen):
            #print(str(j) + "/" + str(colLen))
            workspace[0][j] = workspace[0][j - 1]*self.transProb[j - 1]*self.valProb[j][inputModel[j]]
        for i in range(1, rowLen):
            for j in range(1, colLen):
                calcSpace[0] = workspace[i - 1][j]*self.stayProb[j]*self.valProb[j][inputModel[i + j]]
                calcSpace[1] = workspace[i][j - 1]*self.transProb[j - 1]*self.valProb[j][inputModel[i + j]]
                workspace[i][j] = sum(calcSpace)
```

```
        return workspace[rowLen - 1][colLen - 1]*self.transProb[len(self.transProb) - 1]

    def vitarbi(self, inputModel):
        colLen = self.stateNum
        rowLen = len(inputModel) - colLen + 1
        workspace = [[0 for i in range(colLen)] for j in range(rowLen)]
        #print(str(len(workspace)) + ", " + str(len(workspace[0])))
        #print(workspace[colLen - 1][rowLen - 1])
        calcSpace = [0, 0]
        workspace[0][0] = self.valProb[0][inputModel[0]]
        for i in range(1, rowLen):
            #print(str(i) + "/" + str(rowLen))
            workspace[i][0] = workspace[i - 1][0]*self.stayProb[0]*self.valProb[0][inputModel[i]]
        for j in range(1, colLen):
            #print(str(j) + "/" + str(colLen))
            workspace[0][j] = workspace[0][j - 1]*self.transProb[j - 1]*self.valProb[j][inputModel[j]]
        for i in range(1, rowLen):
            for j in range(1, colLen):
                calcSpace[0] = workspace[i - 1][j]*self.stayProb[j]*self.valProb[j][inputModel[i + j]]
                calcSpace[1] = workspace[i][j - 1]*self.transProb[j - 1]*self.valProb[j][inputModel[i + j]]
                workspace[i][j] = max(calcSpace)
        return workspace[rowLen - 1][colLen - 1]*self.transProb[len(self.transProb) - 1]


    def outputSimu(self):
        output = []
        state = 0
        while state != self.stateNum:
            pick = random.random()
            for i in range(self.valNum):
                if pick < sum(self.valProbList[state][:i + 1]):
                    output.append(self.label[i])
                    break

            #動くかどうするか
            judge = random.random()
            if judge < self.transProb[state]:
                state += 1
        #print(output)
        return output
```

4 実験内容

実験内容はまずソースコード 1 を使用し複数の HMM モデルを作成する．
HMM クラスの output メソッドを使用し結果の系列を排出．これをすべての HMM モデルで forward アルゴリズムと vitabi アルゴリズムを行い認識率を見る．今回は，

1. 排出確率のみを変えて認識
2. 遷移確率のみを変えて認識
3. すべてバラバラで認識

5 結果

5.1 排出確率のみを変えて認識

今回使用したのは，排出は「”Red”, ”Green”, ”Blue”」の三種類でツボは 3 つ遷移確率は順に「0.8, 0.3, 0.5」である．
排出確率は「”Red”, ”Green”, ”Blue”」の順で，

model1 [0.6, 0.3, 0.1], [0.1, 0.6, 0.3], [0.3, 0.1, 0.6]

model2 [0.3, 0.1, 0.6], [0.6, 0.3, 0.1], [0.1, 0.6, 0.3]

model3 [0.1, 0.6, 0.3], [0.3, 0.1, 0.6], [0.6, 0.3, 0.1]

の 3 モデルで実験する．結果は以下の表 1，2 のようになった

表 1: 排出率のみ違う場合の forward アルゴリズムの認識率

	model1	model2	model3
model1	75.12	11.65	13.23
model2	13.99	74.85	11.16
model3	11.71	17.52	70.77

表 2: 排出率のみ違う場合の vitarbi アルゴリズムの認識率

	model1	model2	model3
model1	74.44	13.3	12.26
model2	14.95	73.38	11.67
model3	14.99	18.06	66.95

5.2 遷移確率のみを変えて認識

排出は同じく「”Red”, ”Green”, ”Blue”」の三種類でツボは 3 つ排出確率は「”Red”, ”Green”, ”Blue”」の順で，[0.6, 0.3, 0.1], [0.1, 0.6, 0.3], [0.3, 0.1, 0.6] とした．
遷移確率は

model1 [0.8, 0.3, 0.5]

model2 [0.5, 0.8, 0.3]

model3 [0.3, 0.5, 0.8]

の 3 モデルで実験する．結果は以下の表 3，4 のようになった

表 3: 遷移確率のみ違う場合の forward アルゴリズムの認識率

	model1	model2	model3
model1	48.78	26.05	25.17
model2	20.73	48.57	30.7
model3	22.93	17.28	59.79

表 4: 遷移確率のみ違う場合の vitarbi アルゴリズムの認識率

	model1	model2	model3
model1	51.39	29.19	19.42
model1	20.79	53.19	26.02
model1	26.91	22.79	50.3

5.3 すべて違う場合の認識

すべての確率を変えた場合結果は以下の表 5，6 のようになった

表 5: すべて違う場合の forward アルゴリズムの認識率

	model1	model2	model3
model1	84.97	7.95	7.08
model1	7.16	83.24	9.6
model1	9.53	10.24	80.23

表 6: すべて違う場合の vitarbi アルゴリズムの認識率

	model1	model2	model3
model1	85.32	7.9	6.78
model1	8.25	82.24	9.51
model1	10.66	10.02	79.32

6 考察

以上の結果から HMM によりモデルの推定が行えていることがわかる．
詳細を見た場合認識率が比較的高く出たものから順に「すべてバラバラ」「排出率のみ違う」「遷移確率のみ違う」となった。ここからわかるのは，排出率が異なる場合は出てくるパターンが異なり，遷移確率は出力の長さが異なるので出てくるパターンが異なる方が認識がしやすいのではないかと考えた．また，比較対象にばらつきがあるほど高い認識率がでて似ているほど誤認識が増える．
また，forward アルゴリズムと vitarbi アルゴリズムでは認識率には大きな差は見られなかったが少し forward アルゴリズムのほうが高いものが多かった．これより最大化するルートをとるよりもすべての確立の総和のほうが確実性が増すのではないかと考えた．

次に学習アルゴリズムは尤度関数を最大にするようにラグランジュの未定乗数を持ち出しうまい変形をしていくことで解いていた．このため拘束条件を満たしながら最適化されていく．
ϕ に関しては設定するモデルが離散か連続かなどにより異なり離散の場合微分ができないため総当たりをしていく方法などが考えられる．

7 参考文献

[1] C.M. ビショップ，“パターン認識と機械学習　下”，丸善出版 (2012)

8 英文を読んでの要約

この章では，もう一つの課題である “An Introduction to Hidden Markov Models” を読んだ要約を記す．この文は HMM とは何かというのをコインの問題とツボとボールの問題を用いて説明している．そのうえで HMM の定義に関する以下の三つの問題を提起している．

1. どのように，観測順序を得るのか
2. 隠れた状態の順序の推定法
3. パラメータの最適化方法

以上のようなことから 6 章でも記した通り様々なパターンに応じて適切にパラメータの調整を行うことが大切だと感じた．