パターン認識入門

木曜2限版

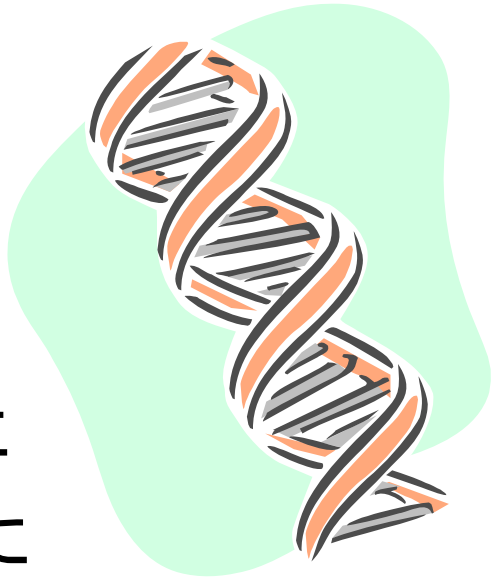
パターン認識

- 音や画像に中に隠れた
  パターンを認識する。
  - 音素・音節・単語・文・・・
  - 基本図形・文字・指紋・物体・人物・顔・・・
- 「パターン」は唯一のデータではなく、
  似通ったデータの集まりを表している。
  - 多様性
  - ノイズ
- 「等しい」から「似ている」へ
- 「～だ」から「～らしい」へ

「等しい」から「似ている」へ

- 完全に等しいかどうかではなく、
  「似ているか」どうかを判定する。
  - パターンを代表する模範的データと
    どのくらい似ているか。
- 例：二つの文字列の比較
  - 多少の文字の食い違いや、
    文字の欠落を許して、似ているか。
  - アラインメントという。
- 動的計画法を活用。

DNAやタンパクの比較



- DNA --- 塩基（四種類）の配列
  - セントラル・ドグマ

    DNA → RNA → タンパク
  - 三塩基（コドン）が一つのアミノ酸に
  - 似ている配列は似ているタンパクに
  - 似ている配列は同じ祖先から進化
- タンパク --- アミノ酸（20種類）の配列
  - 似ている配列は似た構造に
  - 似ている配列は似た機能を

アラインメント

- アラインメント
  二つ(複数)の文字列の比較
  - 音声認識・文字認識
  - DNAやタンパクの比較

GACGGATTAG と GATCGGAATAG

ギャップ　　不一致

GA-CGGATTAG
GATCGGAATAG

アラインメント

- ぴったり同じでなくとも、
  似ているかどうかを判定。
- スコア
  - 一致
  - 不一致
  - ギャップ

  足し合わせる → アラインメントの類似度
- 類似度が最大の
  アラインメント（ギャップの入れ方）を求める。

最適化

アラインメントの最適化:
素朴な方法

S_1, S_2の長さ n, m $(n < m)$

- S_1に$(m - n)$個のギャップを入れた列を作りスコア計算 … ${}_mC_{(m-n)}$通り
- S_1に$(m - n + 1)$個S_2に1個のギャップを入れた列を作りスコア計算 …… ${}_{m+1}C_{(m-n+1)} \cdot {}_{m+1}C_1$通り
- …

S_1 GACGGATTAG

S_2 GATCGGAATAG

```
-GACGGATTAG 9
G-ACGGATTAG 12
GA-CGGATTAG 15
GAC-GGATTAG 12
GACG-GATTAG 9
GACGG-ATTAG 9
GACGGA-TTAG 6
GACGGAT-TAG 9
GACGGATT-AG 6
GACGGATTA-G 3
GACGGATTAG- 0

GATCGGAATAG
```

動的計画法

- プロ野球日本シリーズにおける優勝の確率

P(i, j): Aがシリーズに勝つまでにあとi勝、
Bはあとj勝という状況で、
最終的にAがシリーズに勝つ確率

$P(i, j) = 1$ $i=0$ かつ $j>0$

$= 0$ $i>0$ かつ $j=0$

$= (P(i-1, j) + P(i, j-1))/2$ $i>0$ かつ $j>0$

i →

j ↓

	0	0	0	0	
1	1/2	1/4	1/8		
1	3/4	1/2			
1	7/8				

AとBは
同じ強さと
仮定

動的計画法

- テーブルを用意して、
  再帰的な計算の重複を避ける。
- テーブルの中身を順に埋めることにより、
  求める値を計算する。
- 各種の最適化問題に適用。
  - アラインメント
  - DNA・RNAのエネルギー最小化
  - 構文解析

例

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG と **AAC** をアラインするには

ATA と **AA** のアラインメントに **G** と **C** を付加

2　**ATA** / **A-A**　→　**ATAG** / **A-AC**　1　これを採用！

ATA と **AAC** のアラインメントに **G** と **-** を付加

0　**ATA** / **AAC**　→　**ATAG** / **AAC-**　−2

ATAG と **AA** のアラインメントに **-** と **C** を付加

0　**ATAG** / **A-A-**　→　**ATAG-** / **A-A-C**　−2

アラインメントの動的計画法

- 二つの配列s0とs1の間の類似度
- a[i][j] : s0の部分配列s0[0],...,s0[i−1]と
  s1の部分配列s1[0],...,s1[j−1]の間の類似度
- a[i][j] = max{ a[i][j−1] + g,
  a[i−1][j−1] + q(s0[i−1], s1[j−1]),
  a[i−1][j] + g}

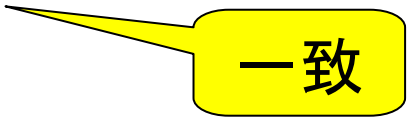
  g : ギャップのペナルティ（負の数）

  q(c, d) : c と d の類似度

  c と d が等しければ適当なスコア（正）
  似ていればそれなりのスコア
  似ていなければ不一致のペナルティ（負）

一文字の類似度を返すメソッドq

```
# 文字cと文字dの類似度を返す
def q(c,d)
    if c == d
        2
    else
        -1
    end
end
```



境界条件

- a[0][0] = 0
- a[0][j] = a[0][j−1] + g （j>0）
- a[i][0] = a[i−1][0] + g （i>0）

例えば g = −2

- 結局

  a[0][j] = j*g

  a[i][0] = i*g

- となる。要するに、ギャップばかり。

スコア:　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC

	s0	**A**	**T**	**A**	**G**
s1	0	−2	−4	−6	−8
A	−2				
A	−4				
C	−6				

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC

	s0	**A**	**T**	**A**	**G**
s1	0	−2	−4	−6	−8
A	−2	2			
A	−4				
C	−6				

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC

	s0	**A**	**T**	**A**	**G**
s1	0	−2	−4	−6	−8
A	−2	2	0		
A	−4	0			
C	−6				

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC

	s0	**A**	**T**	**A**	**G**
s1	0	−2	−4	−6	−8
A	−2	2	0	−2	−4
A	−4	0	1	2	0
C	−6	−2	−1	0	

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC

	s0	**A**	**T**	**A**	**G**
s1	0	−2	−4	−6	−8
A	−2	2	0	−2	−4
A	−4	0	1	2	0
C	−6	−2	−1	0	

ATA
A-A

ATAG
A-A-

ATA
AAC

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC

	s0	**A**	**T**	**A**	**G**
s1	0	−2	−4	−6	−8
A	−2	2	0	−2	−4
A	−4	0	1	2	0
C	−6	−2	−1	0	1

ATA
A-A

ATAG
A-A-

ATA
AAC

ATAG
A-AC

Ruby

トレースバック

- 最大の類似度を与えるアラインメントを提示するために、
  配列の最後から、それまでの選択を振り返る（トレースバック）。

```
m = s0.size
n = s1.size
i = m
j = n
while i>0 && j>0
  if a[i][j] == a[i][j-1] + g
    gap0[i] = gap0[i] + 1
    j = j - 1
  elsif a[i][j] == a[i-1][j-1] + q(s0[i-1],s1[j-1])
    i = i - 1
    j = j - 1
  elsif a[i][j] == a[i-1][j] + g
    gap1[j] = gap1[j] + 1
    i = i - 1
  end
end
```

gap0[i]は、**s0**の**i**番目の文字の前に入るギャップの数。
0で初期化しておく。

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC

	s0	**A**	**T**	**A**	**G**	gap1
s1	0	−2	−4	−6	−8	0
A	−2	2	0	−2	−4	1
A	−4	0	1	2	0	0
C	−6	−2	−1	0	1	0

gap0	0	0	0	0	0

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC

	s0	**A**	**T**	**A**	**G**	gap1
s1	0	−2	−4	−6	−8	0
A	−2	2	0	−2	−4	1
A	−4	0	1	2	0	0
C	−6	−2	−1	0	1	0
gap0	0	0	0	0	0	

ATA
A-A

ATAG
A-AC

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC

	s0	**A**	**T**	**A**	**G**	gap1
s1	0	−2	−4	−6	−8	0
A	−2	2	0	−2	−4	1
A	−4	0	1	2	0	0
C	−6	−2	−1	0	1	0

gap0	0	0	0	0	0

スコア：　一致　+2
不一致　−1
ギャップ　−2

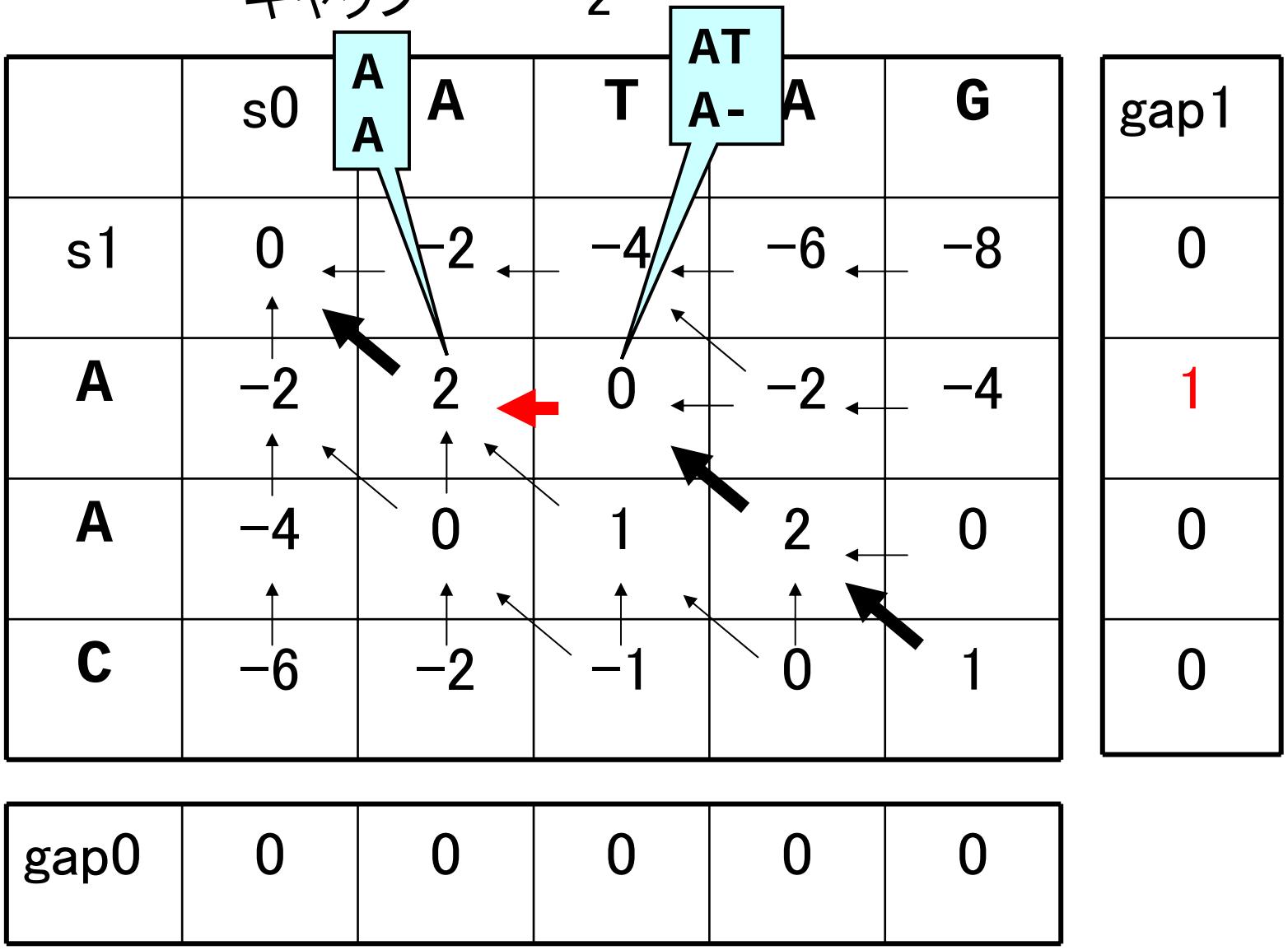
ATAG
A-AC

	s0	**A**	**T**	**A**	**G**	gap1
s1	0	−2	−4	−6	−8	0
A	−2	2	0	−2	−4	1
A	−4	0	1	2	0	0
C	−6	−2	−1	0	1	0

gap0	0	0	0	0	0

スコア：　一致　+2
不一致　−1
ギャップ　−2

ATAG
A-AC



Ruby

トレースバックの表示

- **gap0**と**gap1**を利用し、文字列**s0**と**s1**を
  ギャップを含めて端末に表示する。

  e.g. **ATAG**

  A-AC

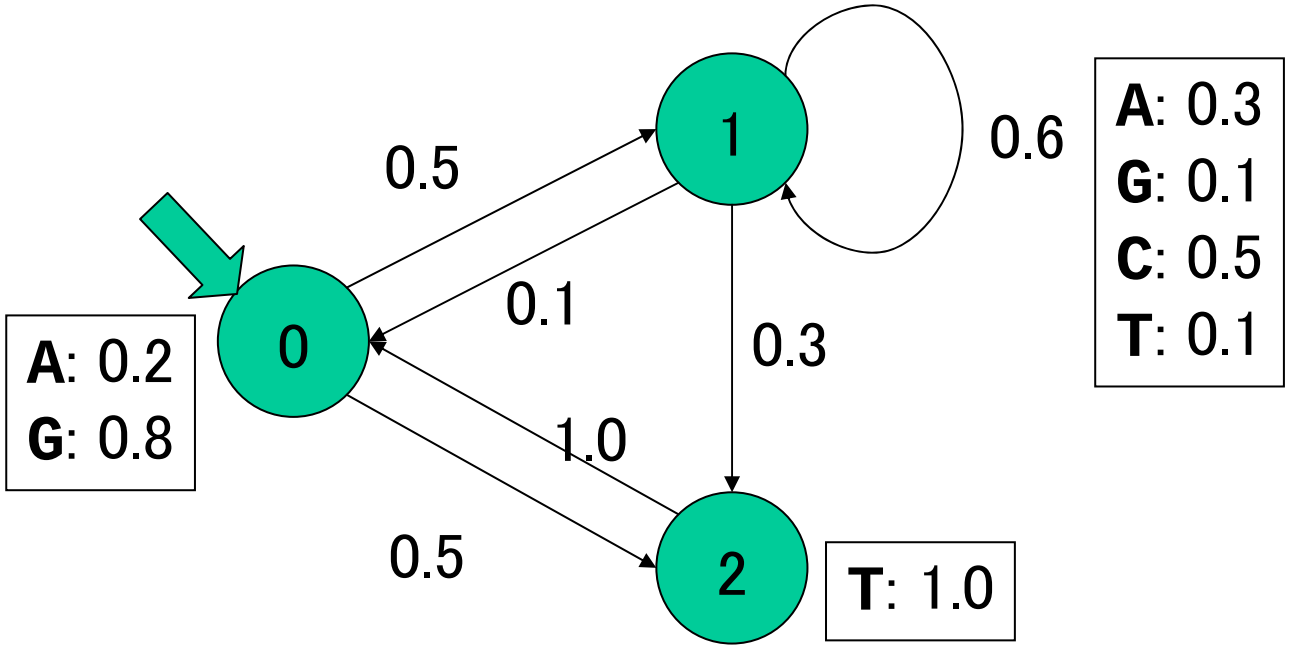
```
for i in 0..m
    for k in 1..gap0[i]
        print("-")
    end
    if i<m
        print(s0[i..i])
    end
end
print "¥n"
```

「～だ」から「～らしい」へ

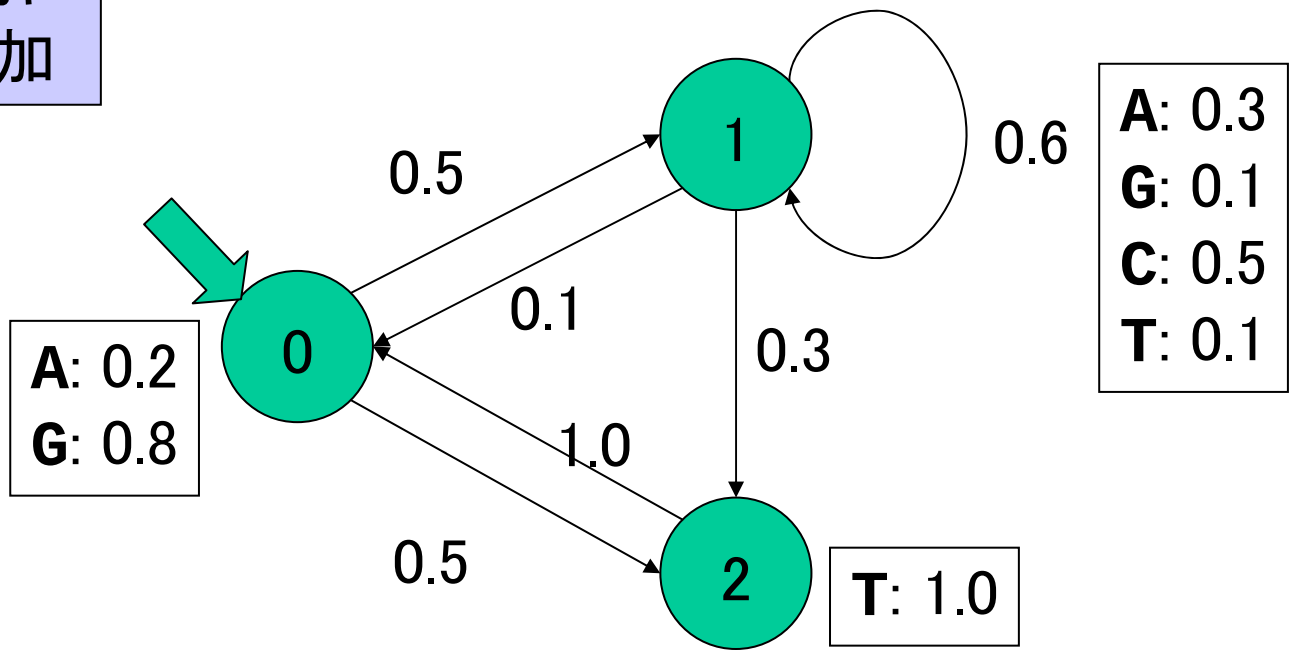
- あるデータがパターンに従っているか、
  否かを、断定するのではなく、
  その確からしさを求める。
- 例：文字列の判別
  - 文字列がパターンに従っている
    確からしさを求める。
  - 隠れマルコフモデル
- ここでも動的計画法を活用。

隠れマルコフモデル

- 有限オートマトンに確率を付加したようなもの。
- 遷移ではなく、状態に文字が対応。
  出力文字と考える。（本質的ではない。）
- 各遷移と各出力文字に確率が与えられる。



遷移と出力に
確率が付加



状態遷移

0 1 0　　0.5*0.1 = 0.05

0 1 1　　0.5*0.6 = 0.30

0 1 2　　0.5*0.3 = 0.15

…

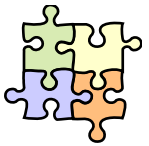
出力文字列

GCG　　0.8*0.5*0.8=0.32 → 0.016

GCG　　0.8*0.5*0.1=0.04 → 0.012

GCT　　0.8*0.5*0.1=0.04 → 0.012

GCT　　0.8*0.5*1.0=0.40 → 0.060



隠れマルコフ過程

- マルコフ仮定
  - 次の状態は、現在の状態のみに依存し、過去の履歴には依存しない。
- 隠れ？
  - 各状態の出力も確率的に定まる。
  - 従って、状態遷移は直接観測できない。
- 観測可能な現象の背後にある確率的なモデル

実際のパターン認識

- 音声認識
  - 波形データから音素を抽出
  - 音素列に対する隠れマルコフモデル
- 手書き文字認識
  - ストロークをセグメントに分割
  - セグメントの列に対するアラインメント
- 画像認識
  - 様々な前処理
    - エッジ検出・背景分別・・・
  - アラインメント・隠れマルコフモデル・・・

実際は
はるかに複雑

実際は
もっと複雑

画像の種類に
応じた様々な
手法