6 장. 질적 분류

오일석, 패턴인식, 교보문고, 2008.

들어가는 말

- 세상에는 참으로 많은 데이터가 있다.
  - 계량 데이터
    - 점수, 매출액, GDP, BOD, 속도, 마찰계수, 토끼 개체수 등
    - 거리 개념 있다. 5는 31보다 크다. 5는 10보다 7에 가깝다.
  - 비계량 데이터
    - 직업, 행정 구역, 혈액형, 성씨, PC 브랜드 등
    - 거리 개념 없다. ‘O형은 B형보다 A형에 가깝다’는 성립 안한다.

들어가는 말

- 6장은 비계량 데이터의 분류를 다룸
  - 질적 분류기
    - 결정 트리 (6.1절)
    - 스트링 인식기 (6.3절)

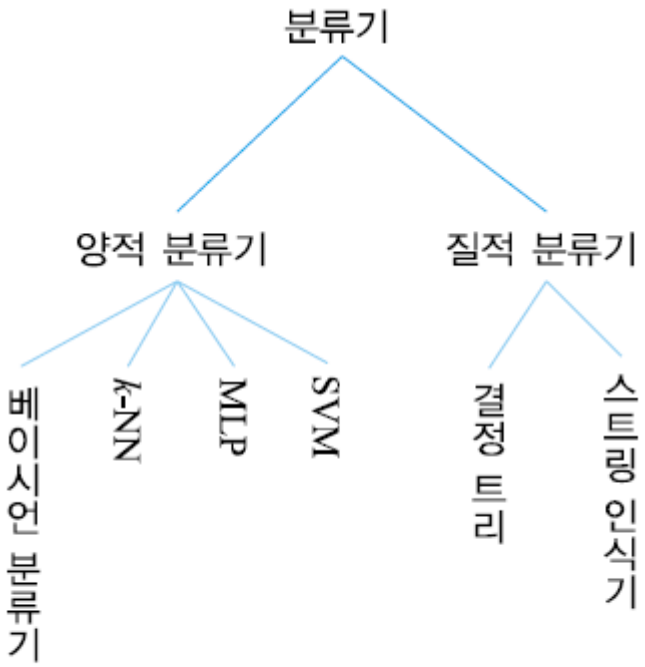


그림 6.1 양적 분류기와 질적 분류기

6.1.1 원리

- 결정 트리의 원리
  - 스무고개와 개념이 비슷
  - 최적 기준에 따라 자동으로 질문을 만들어야 함

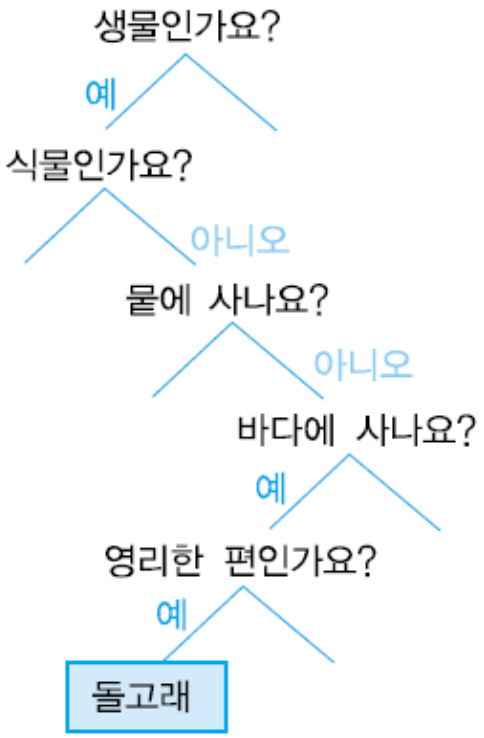
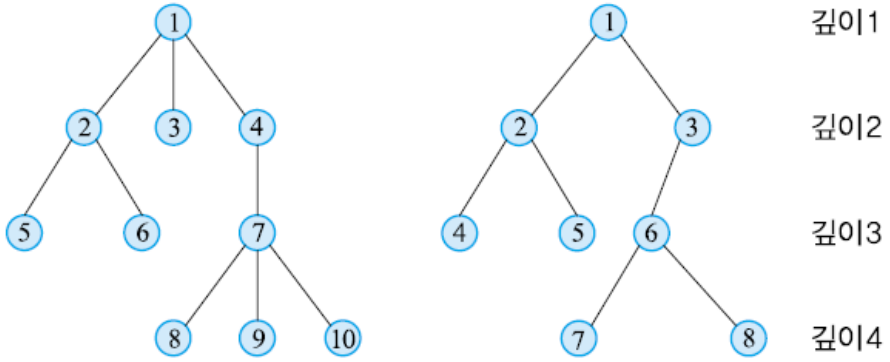


그림 6.2 스무고개 놀이

- 몇 가지 고려 사항
  1. 노드에서 몇 개의 가지로 나눌 것인가?
  2. 각 노드의 질문을 어떻게 만들 것인가?
  3. 언제 멈출 것인가?
  4. 잎 노드를 어느 부류에 할당할 것인가?

6.1.1 원리

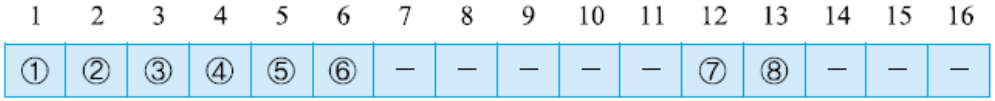
- 결정 트리의 표현
  - 트리 또는 이진 트리 사용



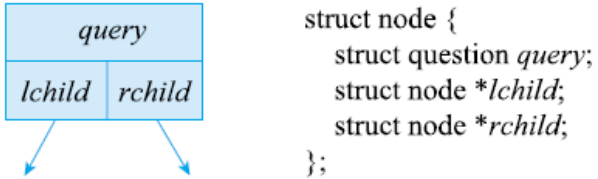
(a) 트리 (b) 이진 트리

그림 6.3 트리와 이진 트리

- 이진 트리의 구현



(a) 1 차원 배열 표현



(b) 연결 리스트 표현

그림 6.4 이진 트리 표현 방법

6.1.2 노드에서의 질문

- 결정 트리의 노드
  - 노드의 분기

$$\left.\begin{aligned} X_{Tleft} \cup X_{Tright} &= X_T \\ X_{Tleft} \cap X_{Tright} &= \varnothing \end{aligned}\right\} \quad (6.1)$$

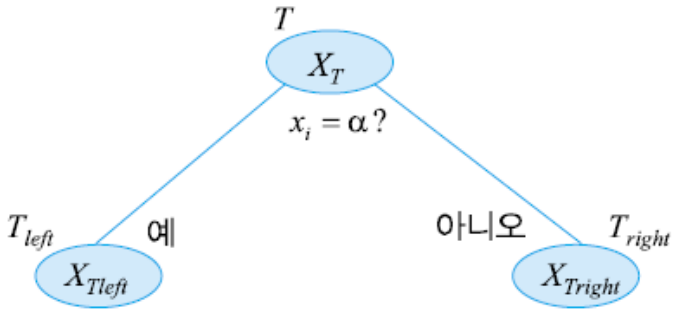


그림 6.5 노드의 분기

  - 질문 x_i=α? 어떻게 만들 것인가?
    - d 개의 특징이 있고 그들이 평균 n 개의 값을 가진다면 dn 개의 후보 질문
    - 그들 중 어느 것을 취해야 가장 유리한가?

6.1.2 노드에서의 질문

- 유리한 정도의 판단 기준은?
  - X_{Tleft}와 X_{Tright}가 동질일 수록 좋다.

- 불순도 측정 기준
  - 엔트로피 $$im(T) = -\sum_{i=1}^{M} P(\omega_i \mid T) \log_2 P(\omega_i \mid T) \quad (6.2)$$
  - 지니 불순도 $$im(T) = 1 - \sum_{i=1}^{M} P(\omega_i \mid T)^2 = \sum_{i \neq j} P(\omega_i \mid T) P(\omega_j \mid T) \quad (6.3)$$
  - 오분류 불순도 $$im(T) = 1 - \max_i P(\omega_i \mid T) \quad (6.4)$$
  - 노드 T에서 ω_i가 발생할 확률은

$$P(\omega_i \mid T) = \frac{X_T\text{에서 } \omega_i\text{에 속한 샘플의 수}}{|X_T|} \quad (6.5)$$

6.1.2 노드에서의 질문

- 예제 6.1 불순도 측정

노드 T의 샘플 집합 X_T가 아래와 같다고 하자.

$$X_T = \{(\mathbf{x}_1,\omega_2),\ (\mathbf{x}_2,\omega_1),\ (\mathbf{x}_3,\omega_3),\ (\mathbf{x}_4,\omega_2),\ (\mathbf{x}_5,\omega_2),\ (\mathbf{x}_6,\omega_2),\ (\mathbf{x}_7,\omega_1),\ (\mathbf{x}_8,\omega_3),\ (\mathbf{x}_9,\omega_1)\}$$

$$P(\omega_1 \mid T) = 3/9,\ P(\omega_2 \mid T) = 4/9,\ P(\omega_3 \mid T) = 2/9$$

엔트로피 불순도: $im(T) = -(\frac{3}{9}\log_2\frac{3}{9} + \frac{4}{9}\log_2\frac{4}{9} + \frac{2}{9}\log_2\frac{2}{9}) = 1.5305$

지니 불순도: $im(T) = 1 - (\frac{3^2}{9^2} + \frac{4^2}{9^2} + \frac{2^2}{9^2}) = 0.642$

오분류 불순도: $im(T) = 1 - \frac{4}{9} = 0.556$

6.1.2 노드에서의 질문

- 노드에서 질문 선택
  - 불순도 감소량 또는 투잉 기준이 최대인 질문을 취함
    - 불순도 감소량

$$\Delta im(T) = im(T) - \frac{|X_{Tleft}|}{|X_T|} im(T_{left}) - \frac{|X_{Tright}|}{|X_T|} im(T_{right}) \quad (6.6)$$

    - 투잉 기준

$$\Delta im(T) = \frac{|X_{Tleft}|}{|X_T|} \frac{|X_{Tright}|}{|X_T|} \left(\sum_{i=1}^{M} |p(\omega_i \mid T_{left}) - p(\omega_i \mid T_{right})| \right)^2 \quad (6.7)$$

6.1.2 노드에서의 질문

- 노드에서 질문 생성
  - 비계량인 경우 $x_i=\alpha$?
  - 계량인 경우 $x_i<\alpha$?
    - 이산
      - 이산 값에 따라 α를 결정
    - 연속
      - 실수 범위를 구간화 하여 α 결정
      - 또는 샘플의 값 분포를 보고 두 값의 가운데를 α로 결정

6.1.2 노드에서의 질문

- 예제 6.2 후보 질문 생성

직업 (x_1): [1,7]의 정수 (1 = 디자이너, 2 = 스포츠맨, 3 = 교수, 4 = 의사, 5 = 공무원, 6 = NGO, 7 = 무직)

선호 품목 (x_2): [1,5]의 정수 (1 = 의류, 2 = 전자 제품, 3 = 스포츠 용품, 4 = 책, 5 = 음식)

몸무게 (x_3): 실수

x_1에 의한 후보 질문: x_1=1?, x_1=2?, x_1=3?, x_1=4?, x_1=5?, x_1=6?, x_1=7?

x_2에 의한 후보 질문: x_2=1?, x_2=2?, x_2=3?, x_2=4?, x_2=5?

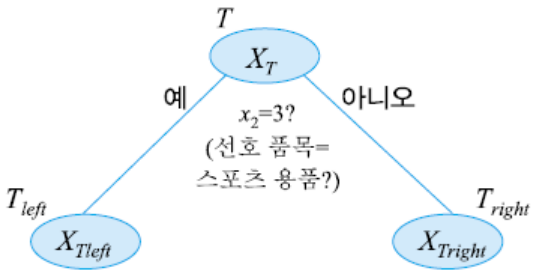
표 6.1에서 x_3의 값의 분포를 조사하면,

45.6, 47.8, 50.6, 65.3, 67.8, 72.8, 88.7, 92.3, 102.2

x_3에 의한 후보 질문: x_3<46.7?, x_3<49.2?, x_3<57.95?, x_3<66.55?, x_3<70.3?, x_3<80.75?, x_3<90.5?, x_3<97.25?

6.1.2 노드에서의 질문

- 예제 6.3 불순도 감소량



$X_T=\{(\mathbf{x}_1,\omega_2), (\mathbf{x}_2,\omega_1), (\mathbf{x}_3,\omega_3), (\mathbf{x}_4,\omega_2), (\mathbf{x}_5,\omega_2), (\mathbf{x}_6,\omega_2), (\mathbf{x}_7,\omega_1), (\mathbf{x}_8,\omega_3), (\mathbf{x}_9,\omega_1)\}\}$

$X_{Tleft}=\{(\mathbf{x}_2,\omega_1), (\mathbf{x}_7,\omega_1), (\mathbf{x}_9,\omega_1)\}$

$X_{Tright}=\{(\mathbf{x}_1,\omega_2), (\mathbf{x}_3,\omega_3), (\mathbf{x}_4,\omega_2), (\mathbf{x}_5,\omega_2), (\mathbf{x}_6,\omega_2), (\mathbf{x}_8,\omega_3)\}$

그림 6.6 '선호 품목=스포츠 용품?' 이라는 질문에 따른 분기 결과

(6.6)의 불순도 감소량 $\Delta im(T) = 0.642 - \frac{3}{9} * 0.0 - \frac{6}{9} * 0.444 = 0.346$

(6.7)의 불순도 감소량 $\Delta im(T) = \frac{3}{9}\frac{6}{9}\left(|1-0| + |0-4/6| + |0-2/6|\right)^2 = 0.889$

6.1.3 학습 알고리즘

- 결정 트리 학습 알고리즘
  - 언제 멈출 것인가?
    - 과적합 vs. 설익은 수렴
  - 잎 노드의 부류 할당

$$\left.\begin{array}{l} T\text{의 부류를 } \omega_k\text{로 한다.} \\ \text{이때 } k = \underset{i}{\operatorname{argmax}}\ P(\omega_i \mid T) \end{array}\right\}$$

알고리즘 [6.1] 결정 트리 학습

입력: 훈련 집합 $X = \{(\mathbf{x}_1, \mathbf{t}_1), \cdots, (\mathbf{x}_N, \mathbf{t}_N)\}$

출력: 결정 트리 R

알고리즘:

```
1. 노드 하나를 생성하고 그것을 R이라 한다.  // 이것이 루트 노드이다.
2. T = R;
3. X_T = X;
4. split_node(T, X_T);    // 루트 노드를 시작점으로 하여 순환 함수를 호출한다.

5. split_node(T, X_T) {   // 순환 함수
6.     노드 T에서 후보 질문을 생성한다.
7.     모든 후보 질문의 불순도 감소량을 측정한다.  // (6.6) 또는 (6.7) 이용
8.     불순도 감소량이 최대인 질문 q를 선택한다.
9.     if (T가 멈춤 조건을 만족) {
10.        T에 부류를 할당한다.
11.        return;
12.    }
13.    else {
14.        q로 X_T를 X_Tleft와 X_Tright로 나눈다.
15.        새로운 노드 T_left와 T_right를 생성한다.
16.        split_node(T_left, X_Tleft);
17.        split_node(T_right, X_Tright);
18.    }
19. }
```

6.1.4 특성

- 결정 트리의 특성
  - 특징 값에 대한 제약이 적다.
    - 계량, 비계량, 혼합 특징을 모두 다룰 수 있다.
    - 특징 전처리 불필요
  - 분류 결과가 '해석 가능'하다.
  - 인식 작업이 매우 빠르다.
  - 가지치기
    - 사전 가지치기
    - 사후 가지치기
  - 불안정성
  - 결정 트리 학습은 욕심 알고리즘
  - 손실 특징을 다루기 쉽다.
    - 대리 분기

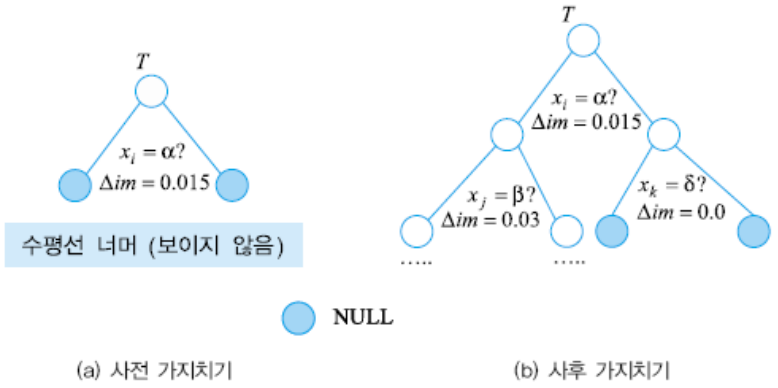


그림 6.7 사전 가지치기와 사후 가지치기

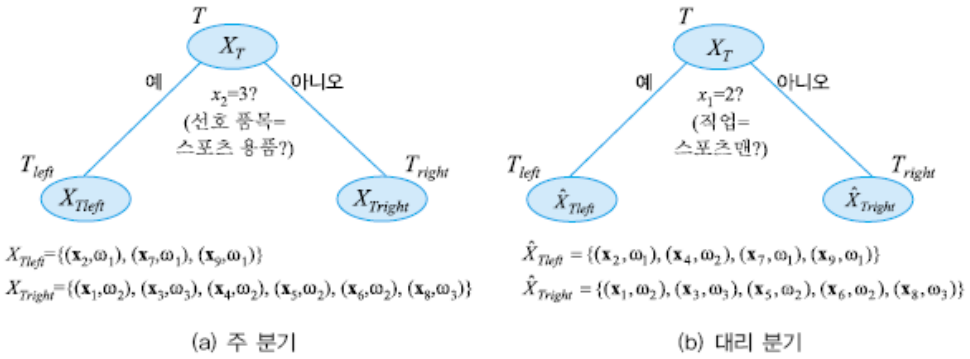


그림 6.8 주 분기와 대리 분기

6.2 CART, ID3, 그리고 C4.5

- 대표적인 결정 트리 시스템 비교

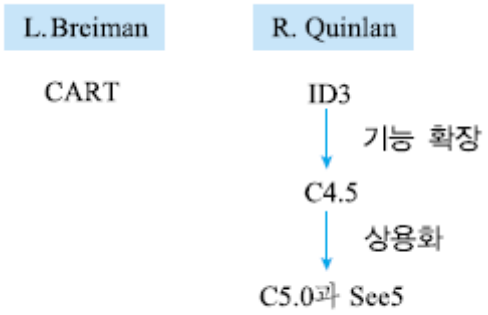


그림 6.9 대표적인 결정 트리의 계통도

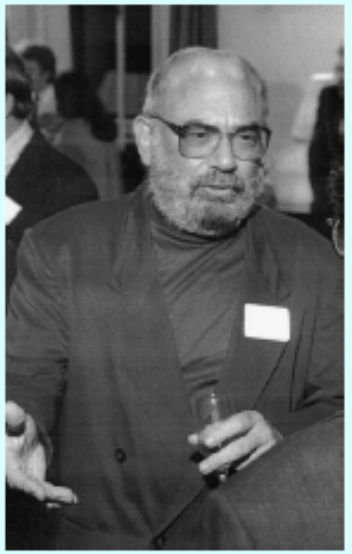
표 6.2 CART, ID3, 그리고 C4.5의 특성 비교

특성	CART	ID3	C4.5
실수 데이터	부등호 질문	등식 질문	부등호 질문
트리 형태	이진 트리	트리	트리
가지치기	잎 노드 병합	x	규칙 집합
분류	지원	지원	지원
회귀	지원	x	x
손실 특징	대리 분기	x	샘플 무시
다중 변수 질문	지원	x	x

- 어느 것이 좋은가?
  - 패턴 인식의 일반적인 질문
  - “어느 것이 다른 것을 지배하지 못하고 어느 것이 다른 것에 지배되지도 않는다.”

Leo Breiman

(1928년 1월 27일 ~ 2005년 7월 7일) 미국



Breiman은 Berkeley 대학의 통계학과 교수이다. 그는 통계학을 컴퓨터 과학과 접목하는데 큰 족적을 남겼다. 특히 패턴 인식 분야에서는 결정 트리 시스템인 CART 그리고 혼합 모델의 일종인 bagging과 random forest로 유명하다. 그는 학생들의 교육에 많은 관심을 기울이고 실천한 사람으로도 평가 받고 있다. Olshen과의 인터뷰 [Olshen01]에서 수학 교육에 대해 '수학을 일상 생활과 관련이 없는 매우 지루하고 끔찍한 것으로 생각하게끔' 한다고 신랄하게 비난하였다. 그리고 실제로 초등 학생을 대상으로 '게임으로 가장한' 방식으로 대수와 기하를 가르치는 열정을 보였다. 또한 연구년을 UNESCO 주선으로 아프리카 대륙의 라이베리아에서 아이들을 가르치기도 하였다.

[Olshen01] Richard Olshen, "A conversation with Leo Breiman," *Statistical Science*, Vol.16, No.2, pp.184-198, 2001.

6.3 스트링 인식기

- 특징 벡터가 가변 길이의 스트링으로 표현되는 응용
  - 궤적을 동서남북 {E, W, S, N}으로 표현하는 경우
  - DNA {A, G, C, T}
  - 온라인 글자의 체인 코드 표현

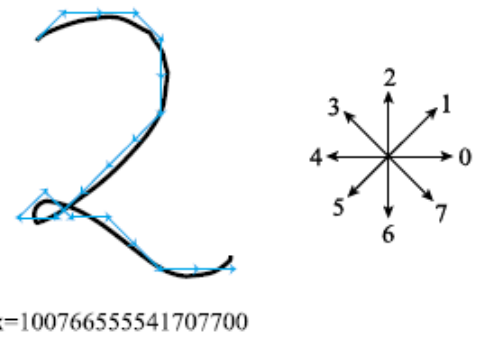


그림 6.10 온라인 필기 숫자의 체인 코드 표현

- 스트링의 거리 계산 방법 필요
  - 거리 정의하면 *k*-NN이나 군집화 등에 적용 가능
  - 거리 정의하더라도 SVM, 신경망에는 적용 불가능
  - 왜?

6.3.1 교정 거리

- 스트링 간의 거리를 어떻게 정의할까?
  - 예) x_1 = aabbacb

    x_2 = abbacb*

    x_3 = aaccabb
  - 해밍 거리가 적절한가?

- 교정 거리edit distance
  - 삽입, 삭제, 대치 등의 연산 비용에 따라 측정하는 거리
  - Levenshtein 거리와 그것의 여러 변종들

6.3.2 Levenshtein 거리

- 표기
  - 테스트 샘플 $\mathbf{x}=x_1x_2\ldots x_c$, 기준 샘플 $\mathbf{y}=y_1y_2\ldots y_r$
  - 삽입과 삭제 비용이 다른 경우 **x**를 **y**로 변환하는 비용과 **y**를 **x**로 변환하는 비용이 다르다.
- Levenshtein 거리가 사용하는 세 가지 연산

1. 삽입 (I): 예를 들어 x_8 다음에 y_9를 삽입하여 **x** = revgniation으로 만든다.
2. 삭제 (D): 예를 들어 x_7을 삭제하여 **x** = revgniton으로 만든다.
3. 대치 (S): 예를 들어 x_3을 y_3으로 대치하여 **x** = recgniaton으로 만든다.

- 예) **x**=revgniaton, **y**=recognition

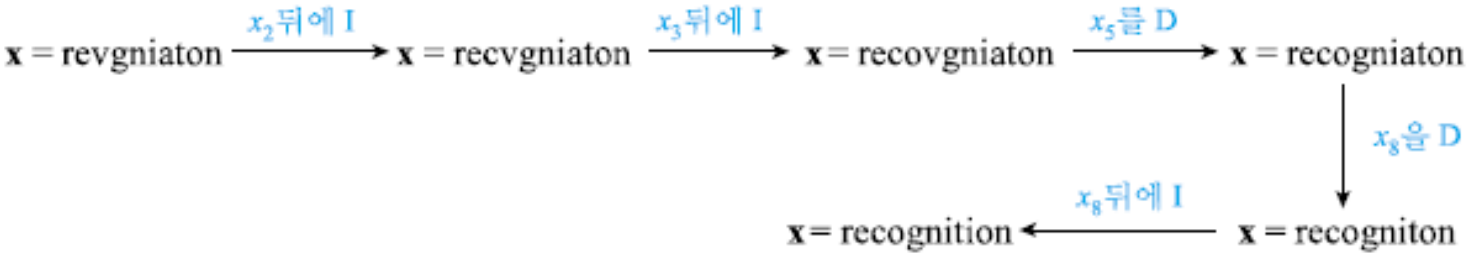
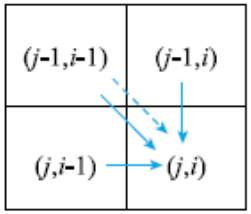


그림 6.11 스트링 교정 사례 (이것보다 더 적은 비용으로 교정이 가능한가?)

6.3.2 Levenshtein 거리

- Levenshtein 거리 계산은 최적화 문제이다.
  - 최소 비용의 변환을 찾아라.
  - 동적 프로그래밍 기법 적용
- 동적 프로그래밍
  - 2차원 배열 D에 그때까지의 최적 거리 기록해 나감
  - $D[j][i]$는 $x_1x_2...x_i$를 $y_1y_2..y_j$로 변환하는데 드는 최소 비용을 가짐
  - $D[r][c]$가 답을 가짐

		0	1	2	3	4	5	6	7	8	9	10
	x =	r	e	v	g	n	i	a	t	o	n	
0	y =	0	1	2	3	4	5	6	7	8	9	10
1	r	1	0	1	2	3	4	5	6	7	8	9
2	e	2	1	0	1	2	3	4	5	6	7	8
3	c	3	2	1	1	2	3	4	5	6	7	8
4	o	4	3	2	2	2	3	4	5	6	6	7
5	g	5	4	3	3	2	3	4	5	6	7	7
6	n	6	5	4	4	3	2	3	4	5	6	7
7	i	7	6	5	5	4	3	2	3	4	5	6
8	t	8	7	6	6	5	4	3	3	3	4	5
9	i	9	8	7	7	6	5	4	4	4	4	5
10	o	10	9	8	8	7	6	5	5	5	4	5
11	n	11	10	9	9	8	7	6	6	6	5	4



변화 없음

대치 (S)

삽입 (I)

삭제 (D)

그림 6.12 Levenshtein 거리 계산

6.3.2 Levenshtein 거리

- Levenshtein 거리 계산 알고리즘
  - 초기화, 전방 계산, 역 추적의 세 단계

알고리즘 [6.3] Levenshtein 거리 계산

입력: 테스트 스트링 **x**와 기준 스트링 **y**

// 삽입과 삭제의 비용이 같으면 대칭이므로 테스트와 기준 스트링의 구별이 불필요

출력: **x**와 **y** 사이의 거리 *d*, 교정 연산 목록 *edit_list*[]

알고리즘:

1. **x**의 길이를 *c*라 하고 **y**의 길이를 *r*이라 한다.
2. (*r*+1)*(*c*+1)의 배열 *D*[0…*r*][0…*c*]를 생성한다.
   // 전방 계산 (라인 3-12)
3. **for** (*i* = 0 **to** *c*) *D*[0][*i*] = *i*; // 행 0의 초기화
4. **for** (*j* = 0 **to** *r*) *D*[*j*][0] = *j*; // 열 0의 초기화
5. **for** (*j* =1 **to** *r*)
6. **for** (*i* = 1 **to** *c*) {
7. **if** ($x_i = y_j$) *scost*=0; **else** *scost*=1; // 대치 비용
8. $D[j][i] = \text{minimum}(\underbrace{D[j-1][i]+1}_{\text{삽입}}, \underbrace{D[j][i-1]+1}_{\text{삭제}}, \underbrace{D[j-1][i-1]+scost}_{\text{대치}})$;
9. *act* = 라인 8에서 최소가 되었던 연산; // 역 추적에 사용할 것임
10. **if** (*act* = '대치' **and** *scost* = 0) *act* = '변화 없음';
11. *action*[*j*][*i*] = *act*;
12. }

6.3.2 Levenshtein 거리

- 세 단계
  - 초기화: 배열 생성
  - 전방 계산: 순환식에 따라 거리 채워 나감
  - 역 추적: 교정 연산 목록 찾음

```
                    // 역 추적에 의한 답 구하기 (라인 13~23)
13. d = D[r][c];   // 표의 맨 오른쪽 아래 요소가 Levenshtein 거리
14. j = r; i = c;   // 맨 오른쪽 아래 요소에서 역추적 시작
15. k = 1;
16. repeat {
17.       if (action[j][i] = '삽입') { edit_list[k] ='삽입'; j--;}
18.       else if (action[j][i] = '삭제') { edit_list[k] ='삭제'; i--;}
19.       else if (action[j][i] = '대치' ) {edit_list[k] ='대치'; j--; i--}
20.       else if (action[j][i] = '변화 없음' ) { edit_list[k] ='변화 없음'; j--; i--}
21.       k++;
22. } until (j = 0 and i = 0);
23. edit_list[.]의 순서를 뒤집어라.
```

- 전방 계산이 사용하는 세 가지 연산

$(j,i-1) \rightarrow (j,i)$: x_i를 삭제한다. 비용 1이 든다. 화살표 →로 표시한다.

$(j-1,i) \rightarrow (j,i)$: y_j를 삽입한다. 비용 1이 든다. 화살표 ↓로 표시한다.

$(j-1,i-1) \rightarrow (j,i)$: $x_i \neq y_j$이면 x_i를 y_j로 대치한다. 비용 1이 든다. 화살표 ↘로 표시한다.

$(j-1,i-1) \rightarrow (j,i)$: $x_i = y_j$이면 아무 변화 없이 이동한다. 비용 0이 든다. 화살표 ↘로 표시한다.

6.3.2 Levenshtein 거리

- 최적 원리에 따른 순환식

  - 최적 원리: $(0,0)$에서 (j,i)까지의 최단 거리는 $(j,i-1)$까지의 최단 거리에 $(j,i-1) \rightarrow (j,i)$의 이동 비용을 더한 값, $(j-1,i)$까지의 최단 거리에 $(j-1,i) \rightarrow (j,i)$의 이동 비용을 더한 값, $(j-1,i-1)$까지의 최단 거리에 $(j-1,i-1) \rightarrow (j,i)$의 이동 비용을 더한 값 중에 가장 작은 값과 같다.

$$\left.\begin{array}{l} D[j][i] = \text{minimum}(\underbrace{D[j-1][i]+1}_{\text{삽입}}, \underbrace{D[j][i-1]+1}_{\text{삭제}}, \underbrace{D[j-1][i-1]+scost}_{\text{대치}}) \\ \text{이때 } x_i = y_j \text{이면 } scost = 0, \text{그렇지 않으면 } scost = 1 \end{array}\right\} \quad (6.9)$$

6.3.2 Levenshtein 거리

- 예제 6.5 Levenshtein 거리 계산
  - **x**=revgniaton, **y**=recognition
  - 몇 가지 계산 예

$$D[1][1] = \text{minimum}(\underbrace{D[0][1]+1}_{\text{삽입}}, \underbrace{D[1][0]+1}_{\text{삭제}}, \underbrace{D[0][0]+0}_{\text{대치}}) = \text{minimum}(1+1, 1+1, \underline{0+0}) = 0$$

action[1][1] = 변화 없음

$$D[1][2] = \text{minimum}(\underbrace{D[0][2]+1}_{\text{삽입}}, \underbrace{D[1][1]+1}_{\text{삭제}}, \underbrace{D[0][1]+1}_{\text{대치}}) = \text{minimum}(2+1, \underline{0+1}, 1+1) = 1$$

action[1][2] = 삭제

$$D[4][3] = \text{minimum}(\underbrace{D[3][3]+1}_{\text{삽입}}, \underbrace{D[4][2]+1}_{\text{삭제}}, \underbrace{D[3][2]+1}_{\text{대치}}) = \text{minimum}(\underline{1+1}, 2+1, \underline{1+1}) = 2$$

action[4][3] = 삽입 또는 대치

6.3.2 Levenshtein 거리

- 몇 가지 특성
  - 교정 연산마다 비용을 다르게 할 수 있다.
  - 삽입과 삭제 비용이 같으면 대칭이다.
  - 대치만 사용하면 해밍 거리가 되고, 삽입과 삭제만 사용하면 최장 공통 부분 스트링 문제가 된다.
  - 계산 복잡도 $\Theta(rc)$

6.3.3 Damerau-Levenshtein 거리

- Damerau-Levenshtein 거리
  - Levenshtein 거리의 확장
  - 교환 연산 추가
  - 예) **x**=pattren, **y**=patternn은 교환 연산 하나로 **x**를 **y**로 교정
  - 철자 교정 등에 유용

6.3.3 Damerau-Levenshtein 거리

- 전방 계산

```
                    // 전방 계산 (라인 3-13)
3.  for (i = 0 to c) D[0][i] = i;   // 행 0의 초기화
4.  for (j = 0 to r) D[j][0] = j;    // 열 0의 초기화
5.  for (j = 1 to r)
6.      for (i = 1 to c) {
7.          if (x_i = y_j) scost = 0; else scost = 1;   // 대치 비용
8.          D[j][i] = minimum(D[j-1][i]+1, D[j][i-1]+1, D[j-1][i-1]+scost);
                              삽입          삭제          대치
9.          if (i>1 and j>1 and x_i = y_{j-1} and x_{i-1} = y_j)
                D[j][i] = minimum(  D[j][i]  , D[j-2][i-2]+scost);
                                  라인 8의 결과        교환
10.         act = 라인 8과 라인 9에서 최소가 되었던 연산;    // 역 추적에 사용
11.         if (act = '대치' and scost = 0) act = '변화 없음';
12.         action[j][i] = act;
13.     }
```

6.3.3 Damerau-Levenshtein 거리

- 역 추적

```
                // 역 추적에 의한 답 구하기
14. d = D[r][c];   // 맨 오른쪽 아래 요소가 Damerau-Levenshtein 거리
15. j = r; i = c;   // 맨 오른쪽 아래 요소에서 역추적 시작
16. k = 1;
17. repeat {
18.       if (action[j][i] = '삽입') { edit_list[k] = '삽입'; j--;}
19.       else if (action[j][i] = '삭제') { edit_list[k] = '삭제'; i--;}
20.       else if (action[j][i] = '대치' ) { edit_list[k] = '대치'; j--; i--}
21.       else if (action[j][i] = '변화 없음' ) { edit_list[k] = '변화 없음'; j--; i--}
22.       else if (action[j][i] = '교환' ) { edit_list[k] = '교환'; j--; j--; i--; i--;}
23.       k++;
24. } until (j = 0 and i = 0);
25. edit_list[.]의 순서를 뒤집어라.
```