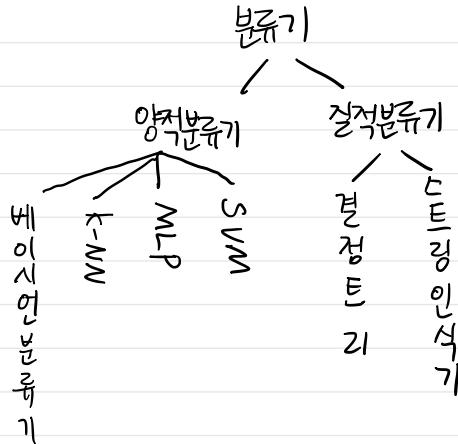


06. 질적분류

데이터  양적 데이터 ex) 점수, GDP, 속도 ...

질적 데이터 ex) 직업, 혈액형, 성씨 ...



6.1 결정트리 (decision tree)

6.1.1 원리

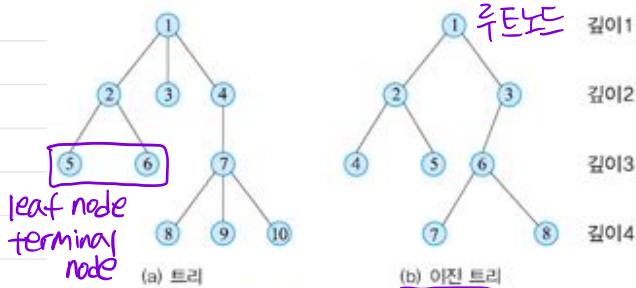


그림 6.3 트리와 이전 트리

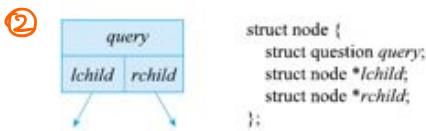
→ 모든 트리는 이전트리로 표현 가능

모든 노드가 최대 두개의 자식을 가짐, 깊이 h에
최대 2^{h-1} 개의 노드를 가짐

이진트리를 표현하는 방식



→ 깊이 h 의 2^{h-1} 개의 노드는
첨자 $2^h, 2^{h-1}+1, \dots, 2^h-1$ 에
저장하면 됨, 노드가 없는 곳은
NULL로 표시



(b) 연결 리스트 표현

그림 6.4 이진 트리 표현 방법

→ 불균형 트리에서는 NULL 요소가
많아 메모리 낭비가 심하므로
linked list 사용

잎 노드에는 부류가 할당되어 있다. 잎 노드에 도달할 때까지
recursively (순환적으로) 반복한다. 잎 노드에 도달하면 그를 그 잎
노드에 해당하는 부류로 분류하고 끝낸다.

* 몇 가지 고려사항

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?

6.1.2 노드에서의 질문

• 결정 트리의 노드

- 노드의 분기

$$X_{T\text{ left}} \cup X_{T\text{ right}} = X_T$$

$$X_{T\text{ left}} \cap X_{T\text{ right}} = \emptyset$$

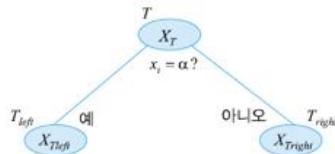


그림 6.5 노드의 분기

- 질문 $x_i = a$? 어떻게 만들 것인가?

d개의 특징이 있고, 그들이 평균 n개의 값을 가진다면

dn개의 후보 질문, 그들 중 어느것을 취해야 가장 좋을까?

후보 질문을 평가하기 위한 기준 함수를 만들어 보자.

- 불순도 (impurity) 측정기준 (n개의 부류가 같은 확률을 가질 때 가장 큰 값은 가정, 불순도가 가장 큼)

- 엔트로피 $im(T) = -\sum_{i=1}^m p(w_i|T) \log_2 p(w_i|T)$ (6.2)

- 지니불순도 $im(T) = 1 - \sum_{i=1}^m p(w_i|T)^2 = \sum_{i \neq j} p(w_i|T)p(w_j|T)$ (6.3)

- 오분류불순도 $im(T) = 1 - \max_i p(w_i|T)$ (6.4)

- 노드 T에서 w_i 가 발생할 확률은

$$p(w_i|T) = \frac{\text{X}_T \text{에서 } w_i \text{에 속한 샘플의 수}}{|\mathcal{X}_T|} \quad (6.5)$$

* 엔트로피

$P(\text{내일=클비} | \text{오늘=맑음})$ 이, $P(\text{내일=맑음} | \text{오늘=맑음})$ 보다 더 유용한 정보이다. 이렇게 어떤 사건의 확률과 그것이 전달하는 정보량은 반비례 관계이다. 이러한 정보량을 나타내는 함수가

$$h(x) = -\log_2 P(x)$$

엔트로피는 랜덤 변수 x 가 가질 수 있는 모든 값(시건)에 대해 정보량(자기 정보)을 평균한 것.

$$H(X) = -\sum_x P(x) \log_2 P(x)$$

$$H(X) = \int_{-\infty}^{\infty} P(x) \log_2 P(x) dx$$

예제 6.1 불순도 측정

노드 T 의 샘플 집합 X_T 가 아래와 같다고 하자.

$$X_T = \{(x_1, w_2), (x_2, w_1), (x_3, w_3), (x_4, w_2), (x_5, w_2), (x_6, w_2), (x_7, w_1), (x_8, w_3), (x_9, w_1)\}$$

$$P(w_1 | T) = \frac{3}{9}, P(w_2 | T) = \frac{4}{9}, P(w_3 | T) = \frac{2}{9}$$

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

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

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

불순도는 하나의 노드를 대상으로 측정한다. 이제 이것을 이용해 그림 6.5의 노드 분기에 필요한 최적의 질문을 고르는데 사용할 기준 함수를 만들어 보자.

분기 결과로 만들어지는 새로운 샘플 집합 X_{Left} 와 X_{Right} 는 가급적 불순도가 낮은 것이 좋다. 따라서 불순도 감소량을 (6.6)과 같이 정의하고 그것을 기준 함수로 사용하면 된다.

• 불순도 감소량

$$\Delta \text{im}(T) = \text{im}(T) - \frac{|X_{\text{Left}}|}{|X_T|} \text{im}(T_{\text{Left}}) - \frac{|X_{\text{Right}}|}{|X_T|} \text{im}(T_{\text{Right}}) \quad (6.6)$$

부모노드 자식노드

• 두 임기준

$$\Delta \text{im}(T) = \frac{|X_{\text{Left}}|}{|X_T|} \frac{|X_{\text{Right}}|}{|X_T|} \left(\sum_{i=1}^M |P(w_i | T_{\text{Left}}) - P(w_i | T_{\text{Right}})| \right)^2 \quad (6.7)$$

후보 질문을 평가하는 기준 함수는 준비되었다. 그렇다면 이제 후보 질문을 어떻게 생성할 것인가?

\leftarrow 비계량인 경우 $x_i = a$?

계량인 경우 $x_i < a$?

이산 이산값에 따라 a 를 결정

연속 실수범위를 구간화하여 a 결정 or
 샘플의 두 값의 가운데를 a 로 결정

예제 6.2 후보 질문 생성

작업 (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)

불문제 (x_3): 실수

x_1 에 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_2 에 의한 후보 질문: $x_2=1?$, $x_2=2?$, $x_2=3?$, $x_2=4?$, $x_2=5?$

표 6.1에서 x_3 의 값의 분포를 조사하면,

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

x_3 에 의한 후보 질문: $x_3 < 46.77?$, $x_3 < 49.22?$, $x_3 < 57.95?$, $x_3 < 66.55?$, $x_3 < 70.37?$,

$x_3 < 80.75?$, $x_3 < 90.52?$, $x_3 < 97.25?$ 두 값의 가운데 설정

총 26개의
후보

예제 6.3 불순도 감소량

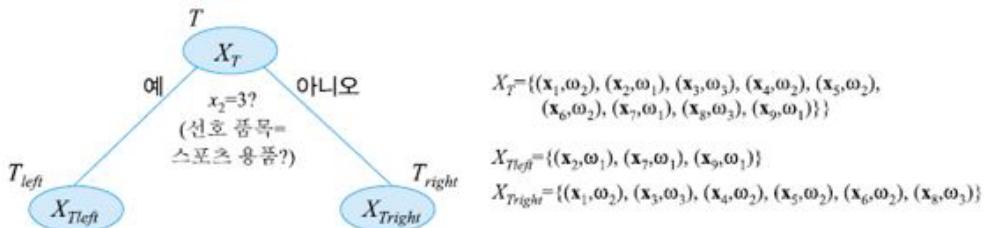


그림 6.6 '선호 품목=스포츠 용품?'이라는 질문에 따른 분기 결과

$$im(T) = 1 - \sum_{i=1}^M P(W_i | T)^2 = 1 - \left(\frac{3}{9^2} + \frac{4}{9^2} + \frac{2}{9^2} \right) = 0.642$$

(지나 불순도 사용)

$$\begin{aligned} (6.6) \text{의 불순도 감소량 } \Delta im(T) &= 0.642 - \frac{3}{9} im(T_{left}) - \frac{6}{9} im(T_{right}) \\ &\leq 0.642 - \frac{3}{9} \left(1 - \left(\frac{2}{3} \right)^2 \right) - \frac{6}{9} \left(1 - \left(\frac{2}{3} \right)^2 + \left(\frac{1}{3} \right)^2 \right) = 0.346 \end{aligned}$$

(6.7)의 불순도 감소량

$$\Delta \text{im}(T) = \frac{3}{9} \cdot \frac{6}{9} \left(|1 - 0| + |0 - \frac{4}{6}| + |0 - \frac{2}{6}| \right)^2 = 0.889$$

여기서는 후보 질문 중 하나에 대해서만 불순도 감소량을 계산하였으나,
실제로는 20개의 후보 모두에 대해 계산 후, 가장 큰 값을 갖는
질문을 택하여 노드 T에 할당하면 된다.

어떤 측정 방법을 쓰는 것이 좋을까?

결정 트리의 성능은 불순도 측정 방법보다 엄축 조건과 노드 가지치기
방법에 더 영향을 받으로 큰 고민하지 않아도 된다.

6.1.3 학습

- 엄축 조건 : 불순도가 충분히 낮을 때 엄주면 된다.

1. 불순도가 0이다.

2. X_T 의 샘플 개수가 임계값 이하

3. 라인 8에서 결정한 9의

불순도 감소량이 임계값 이하

조건 1만 가지고 엄축 결정

→ overfitting / (일반화 능력 ↓)

조건 2, 3 → 임계값 너무 크게 하여

느슨한 조건 사용하면 설득은 수령

(premature convergence)에

도달할 수 있음

- 앞 노드의 부류 할당

T의 부류를 w_k 로 한다

이때 $k = \arg \max_i p(w_i | T)$

알고리즘 [6.1] 결정 트리 학습

입력: 훈련 집합 $X = \{(x_1, t_1), \dots, (x_N, t_N)\}$

출력: 결정 트리 R

알고리즘:

1. 노드 하나를 생성하고 그것을 R 이라 한다. // 이것이 루트 노드이다.
2. $T = R$;
3. $X_T = X$;
4. $\text{split_node}(T, X_T)$; // 루트 노드를 시작점으로 하여 순환 함수를 호출한다.
5. $\text{split_node}(T, X_T)$ { // 순환 함수
6. 노드 T 에서 후보 질문을 생성한다.
7. 모든 후보 질문의 불순도 감소량을 측정한다. // (6.6) 또는 (6.7) 이용
8. 불순도 감소량이 최대인 질문 q 를 선택한다.
9. if (T 가 엄축 조건을 만족) {
10. T 에 부류를 할당한다.
11. return;
12. }
13. else {
14. q 로 X_T 를 $X_{T\text{left}}$ 와 $X_{T\text{right}}$ 로 나눈다.
15. 새로운 노드 T_{left} 와 T_{right} 를 생성한다.
16. $\text{split_node}(T_{\text{left}}, X_{T\text{left}})$;
17. $\text{split_node}(T_{\text{right}}, X_{T\text{right}})$;
18. }
19. }

6.1.4 특성

1. 특징값에 대한 제약이 적다.

- 계량, 비계량, 혼합 특징 모두 다를 수 있음
- 특징 전처리 불필요

2. 분류 결과가 '해석 가능하다'

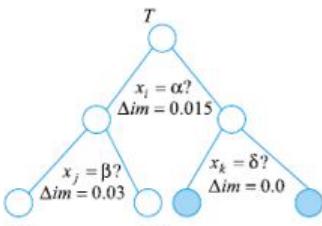
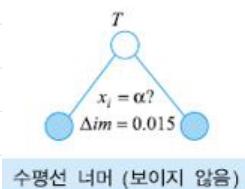
- 루트부터 잎 노드까지 거쳐온 경로 확인
- 고객 관리와 같은 응용에서 매우 유용

3. 학습이 끝난 트리를 가지고 인식하는 작업은 매우 빠름

- 트리의 깊이가 h 일 때, 최대 $h-1$ 번의 노드만 거치면 됨

4. 가지치기 (Pruning)

- 사전 가지치기 : 멈춤 조건 사용하여 미리 멈춤, 멈춤 이후에 어떤 일이 벌어질지 볼 수 없는 수평선 효과가 나타남
- 사후 가지치기 : 임계값 설정을 낮게 허용하거나 불순도 감소량이 0이 될 때까지 분기를 허용하여 충분히 큰 트리를 만든 후, 밑에서부터 잎 노드를 탐치는 연산 수행



(a) 사전 가지치기

(b) 사후 가지치기

그림 6.7 사전 가지치기와 사후 가지치기

5. 불안정성(inStability)

단지 한두 개 샘플의 값이 조금 바뀌었는데 전체 트리 모양이 크게 바뀌는 현상 (1이상이다 아니나니..) but, 혼합 모델의 배깅과 같은 기법에서는 오히려 불안정성을 이용하여 좋은 효과를 얻기도 함.

6. 결정 트리의 학습 알고리즘은 *greedy algorithm*이다.

노드 T는 자기가 처한 상황에만 의존하여 의사 결정을 한다. 이전에 어떤 결정이 있었는지, 이후에 어떤 상황이 벌어질지를 전혀 고려하지 않는다. 따라서 학습 결과로 얻은 트리보다 블교도 측면에서 더 좋으면서 더 짧은 트리가 있을 수 있다.

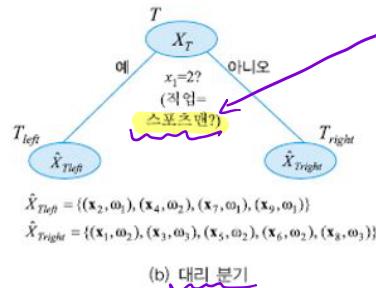
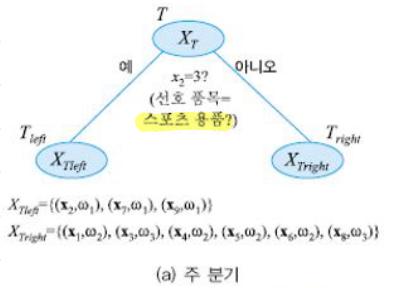
7. 특징을 여러가지 함께 사용 가능.

ex) $3x_1 - 2x_2 < 0.7, x_1 = \alpha$ 이고 $x_2 = \beta$?

8. 손실 특징을 다루기가 다른 분류기에 비해 쉽다.

한 예로 예제 6.2에서 어떤 샘플이 $x = (3, ?, 65.5)$ 이면,

대리 분기 (surrogate split)을 사용할 수 있다. 예로 확인해보자.



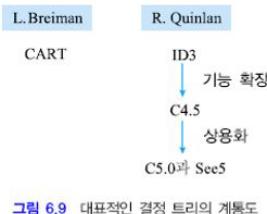
주 분기와 결고가
유사할 질문을
택해야 한다.

X_{Tleft}, X_{Tright} 와 $\hat{X}_{Tleft}, \hat{X}_{Tright}$ 가 유사하도록.

그림 6.8 주 분기와 대리 분기

6.2 CART, ID3, 그리고 C4.5

■ 대표적인 결정 트리 시스템 비교

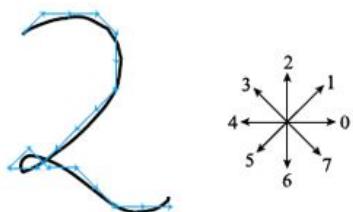


특성	CART	ID3	C4.5
실수 데이터	부등호 질문	등식 질문	부등호 질문
트리 형태	이진 트리	트리	트리
가지치기	잎 노드 병합	x	규칙 합침
분류	지원	지원	지원
회귀	지원	x	x
손실 특징	대리 분기	x	샘플 무시
다중 변수 질문	지원	x	x

“어느 것이 다른 것을 지배하지 못하고 어느 것이 다른 것에
지배되지도 않는다.”

6.3 스트링 인식기

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



x=100766555541707700

그림 6.10 온라인 필기 숫자의 체인 코드 표현

- 스트링의 거리 계산 방법 필요
 - 거리 정의하면 K-NN이나 근접화에 적용 가능
 - 거리 정의하더라도 SVM, 신경망에는 적용 불가
 - 특징값 그 자체를 요구하므로

- 스트링 간의 거리를 어떻게 정의할까?

$$\text{ex) } x_1 = aabbacb$$

$$x_2 = abbaacb*$$

$$x_3 = aaccabbb$$

* 해밍거리

$$x_1 \Rightarrow 10101111 \Rightarrow \text{해밍거리: } 5$$

$$x_3 \Rightarrow 00110100 \Rightarrow \text{해밍거리: } 3$$

* 교정거리

- 삽입, 삭제, 대치 등의 연산 비용에 따라 측정하는 거리

- Levenshtein 거리와 그것의 여러 변종들이 있음

6.3.2 Levenshtein 거리

· 표기

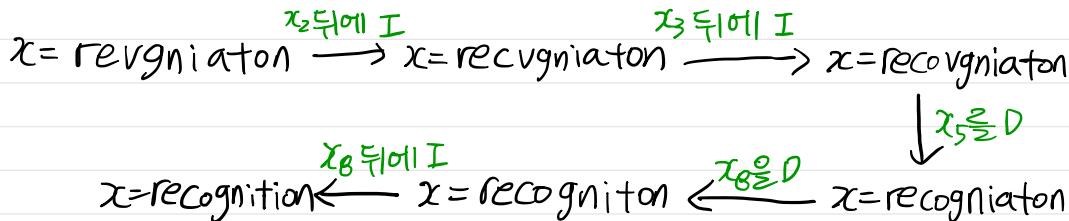
- 테스트 샘플 $x = x_1 x_2 \dots x_c$, 기준 샘플 $y = y_1 y_2 \dots y_r$

- 삽입과 삭제의 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_9 를 삽입하여 $x=revgniation$
2. 삭제(D): 예를 들어 x_7 을 삭제하여 $x=revgniton$
3. 대치(S): 예를 들어 x_3 을 y_3 으로 대치하여 $x=recogniaton$

Ex) $x=revgniatton$, $y=recognition$



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

그림 6.12 Levenshtein 거리 계산

($j-1, i-1$)	($j, i-1$)
($j-1, i$)	(j, i)



• Levenshtein 거리
계산은 최적화 문제이다.

- 최소 비용의 변환을 찾아라
- 동적 프로그래밍 기법

- 동적 프로그래밍
- 2차원 배열 D에
그때까지의 최적 거리
기록해나감

- $D[j][i]$ 는 $x_1x_2\cdots x_i$ 를
 $y_1y_2\cdots y_j$ 로 변환하는데 드는
최소 비용을 가짐

- $(j, i-1) \rightarrow (j, i)$: x_i 삭제, 비용 1, 화살표 \searrow 로 표시
- $(j-1, i) \rightarrow (j, i)$: y_j 삽입, 비용 1, 화살표 \downarrow 로 표시
- $(j-1, i-1) \rightarrow (j, i)$: $x_i \neq y_j$ 이면 x_i 를 y_j 로 대치,
비용 1, 화살표 \swarrow 로 표시
- $(j-1, i-1) \rightarrow (j, i)$: $x_i = y_j$ 이면 아무 변화 없이 이동.

입력: 테스트 스트링 x 와 기준 스트링 y

// 삽입과 삭제의 비용이 같으면 대칭이므로 테스트와 기준 스트링의 구별
이 불필요

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

알고리즘:

배열생성

- x 의 길이를 c 라 하고 y 의 길이를 r 이라 한다.
- $(r+1) * (c+1)$ 의 배열 $D[0 \dots r][0 \dots c]$ 를 생성한다.

// 전방 계산 (라인 3-12)

```

3. for (i=0 to c) D[0][i] = i; // 행 0의 초기화 → 계속 삭제
4. for (j=0 to r) D[j][0] = j; // 열 0의 초기화 → 계속 삽입
5. for (j=1 to r)
6.   for (i=1 to c) { 변화X
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.   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. }
```

// 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 계산

역추적

- 최적 원리에 따른 순환식

- **최적 원리**: $(0, 0)$ 에서 (j, i) 까지의 최단거리는 $(j, i-1)$ 까지의
최단거리에 $(j, i-1) \rightarrow (j, i)$ 의 이동 비용을 더한 값, $(j-1, i)$ 까지의
최단거리에 $(j-1, i) \rightarrow (j, i)$ 의 이동 비용을 더한 값, $(j+1, i-1)$ 까지의
최단거리에 $(j+1, i-1) \rightarrow (j, i)$ 의 이동 비용을 더한 값 중에 가장
작은 값과 같다.

$$D[j][i] = \min_{\text{삽입}} (\underline{D[j-1][i]+1}, \underline{D[j][i-1]+1}, \underline{D[j+1][i-1]+5cost})$$

i 때, $x_i = y_i$ 이면 $5cost = 0$, 그렇지 않으면 1

↳ 여기서는 삽입, 삭제, 대치 모두 같은 비용을 가짐

예제 6.5 Levenshtein 거리 계산

$$\begin{aligned} D[1][1] &= \min_{\text{minimum}} (D[0][1]+1, D[1][0]+1, D[0][0]+0) \\ &= \min_{\text{minimum}} (1+1, 1+1, 0+0) = 0 \end{aligned}$$

action[1][1] = 변화없음

$$\begin{aligned} D[1][2] &= \min_{\text{minimum}} (D[0][2]+1, D[1][1]+1, D[0][1]+1) \\ &= \min_{\text{minimum}} (2+1, 0+1, 1+1) = 1 \end{aligned}$$

$$\begin{aligned} D[4][3] &= \min_{\text{minimum}} (D[3][3]+1, D[4][2]+1, D[3][2]+1) \\ &= \min_{\text{minimum}} (\underline{1+1}, \underline{2+1}, \underline{1+1}) = 2 \end{aligned}$$

어느것을 선택해도 좋다.

- 몇 가지 특성

1. 교정 연산마다 비용을 다르게 할 수 있다. 예를 들어, 'a'를 삽입할 때와, 'b'를 삽입할 때의 비용을 다르게 할 수 있다.
2. 삽입과 삭제 비용이 같으면 대칭이다.
3. 대치만 사용하면 해밍 거리가 되고, 삽입과 삭제만 사용하면 최장 공통 부분 스트링 (Longest Common Substring) 문제가 된다.
4. $r * c$ 개의 요소 값을 구해야 하고, 요소 값을 구하는 데는 고정된 개수의 덱셈과 비교뿐이므로 상수로 볼 수 있다. 따라서 계산복잡도는 $\Theta(r c)$ 이다.

6.3.3 Damerau-Levenshtein 거리

- Levenshtein 거리의 확장으로 교환 연산이 추가되었다.
- ex) $x = \text{Patt}re\text{n}$, $y = \text{Patt}er\text{n}$ 은 교환연산 하나로 z 를 서로 교정 - 철자 교정 등에 유용

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);
9.       삽입           삭제           대치
9.     if (i>1 and j>1 and  $x_i = y_{j-1}$  and  $x_{i-1} = y_j$ )
10.      D[j][i] = minimum(D[j][i], D[j-2][i-2] + scost);
11.        라인 8의 결과           교환
12.      act = 라인 8과 라인 9에서 최소가 되었던 연산; // 역 추적에 사용
13.      if (act = '대치' and scost = 0) act = '변화 없음';
14.      action[j][i] = act;
15.    }
16.  }
```

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.
```