기계학습 개론

- Feature selection (특징 선택)

정보통신공학과

Prof. Jinkyu Kang

MYONGJI UNIVERSITY

이번주 수업의 목차

- 특징 선택
  - 특징의 분별력
  - 특징 선택 문제의 이해
  - 탐색 알고리즘

들어가는 말

- 특징 선택
  - 원래 특징 벡터 $\mathbf{x}_{\text{org}}=(x_1,x_2,\cdots,x_D)$에서 쓸모없거나 중복성이 강한 특징을 찾아 제거하는 작업
  - 차원을 낮추어 주므로 계산 속도 향상 그리고 일반화 능력 증대 효과

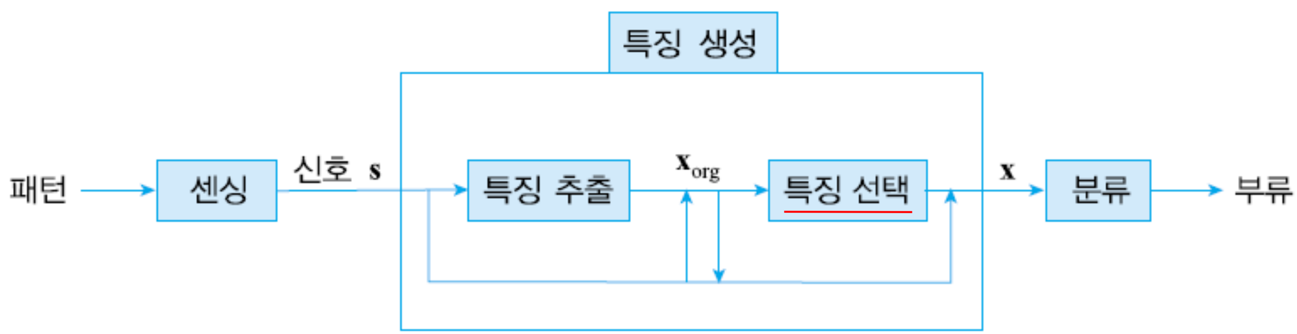
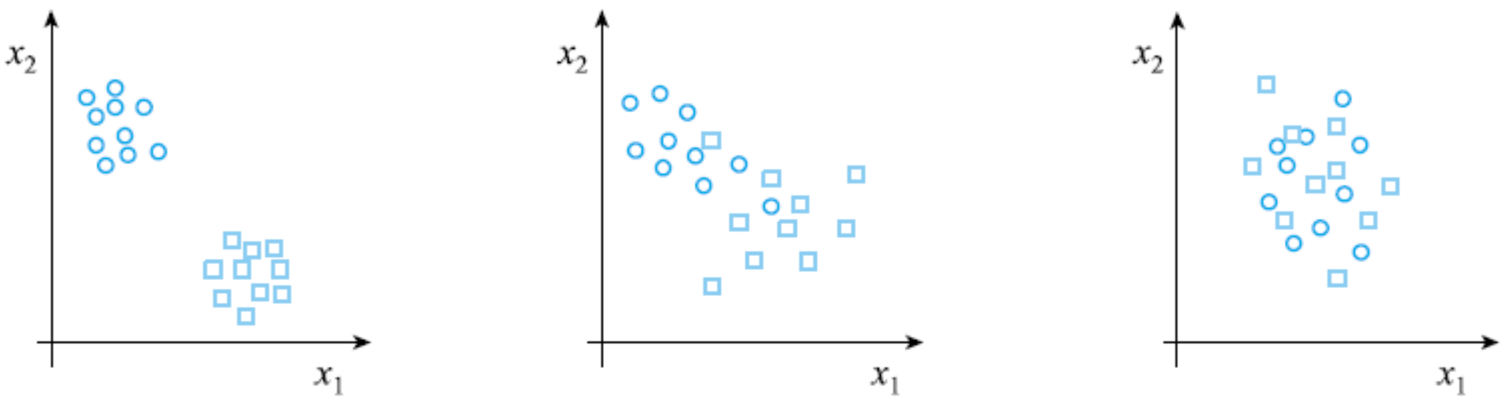


그림 8.2 특징 생성의 절차

특징의 분별력 | 직관적 이해



(a) 부류내 분산은 작고 부류간 분산은 큰 상황

(b) 부류내 분산은 크고 부류간 분산은 작은 상황

(c) 부류간 분산이 0에 가까워 분별력이 없는 상황

그림 9.1 특징 벡터 $\mathbf{x}=(x_1,x_2)^T$의 분별력

- 부류내 분산과 부류간 분산
  - 부류내 분산within-class variance: 같은 부류에 속하는 샘플이 얼마나 퍼져있는지 척도
  - 부류간 분산between-class variance: 서로 다른 부류의 분포가 얼마나 떨어져 있는지 척도
- 부류내 분산은 작고 부류간 분산은 크게 해줄수록 좋은 특징 벡터

MYONGJI UNIVERSITY

특징의 분별력 | 직관적 이해

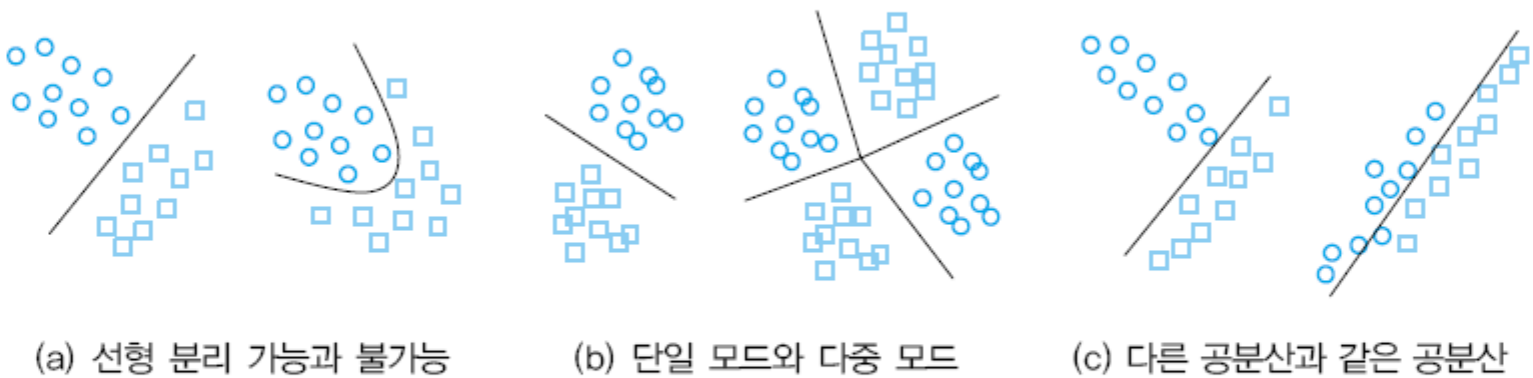


그림 9.2 샘플 분포가 만드는 여러 가지 모양

- 매우 높은 차원의 데이터는 특징 공간의 시각적 표현이 어려움
  - 얼굴인식: 수십 차원의 특징 벡터 / 필기 문자 인식: 수백 차원의 특징 벡터

특징의 분별력 | 분별력 측정

- 분별력 (discriminatory power) / 부류 분리 (class separation)
  - 특징 벡터의 좋고 나쁨을 수량적으로 평가 할 수 있는 기준 함수 필요
  - 사용 되는 척도
    - 다이버전스
    - 훈련 샘플의 거리
    - 분류기 성능

MYONGJI UNIVERSITY

특징의 분별력 | 분별력 측정

- 다이버전스
  - 확률 분포 간의 거리를 이용 (거리를 멀게 해주는 특징일수록 좋다.)

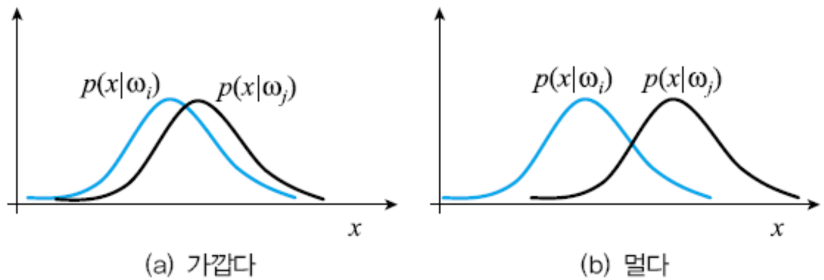


그림 9.3 두 확률 분포 간의 거리

  - 거리 측정은 어떻게 하나? KL 다이버전스 활용

$$d_{ij} = KL(p(\mathbf{x}|\omega_i), p(\mathbf{x}|\omega_j)) + \underline{KL(p(\mathbf{x}|\omega_j), p(\mathbf{x}|\omega_i))} \qquad (9.1) \quad 2\text{ 부류}$$

$$KL(P \parallel Q) = \sum_x P(x)\log_2\frac{P(x)}{Q(x)}$$

$$d = \sum_{i=1}^{M}\sum_{j=1}^{M} P(\omega_i)P(\omega_j)d_{ij} \qquad (9.2) \quad M\text{ 부류}$$

  - 현실적 문제: 확률 분포를 알 때 적용 가능
    - 확률 추정이 안고 있는 차원의 저주 같은 문제를 이어받는다.

특징의 분별력 | 분별력 측정

- 훈련 샘플의 거리
  - 훈련 집합에 있는 샘플을 가지고 직접 측정 (현실 적용이 쉽다.)

$$d_{ij} = \frac{1}{N_i N_j} \sum_{k=1}^{N_i} \sum_{m=1}^{N_j} dist(\mathbf{x}_i^k, \mathbf{x}_j^m) \qquad (9.3)$$

  - $\mathbf{x}_i^k$는 부류 ω_i의 k 번째 샘플
  - $dist(\mathbf{x}_i^k, \mathbf{x}_j^m)$은 두 샘플 간의 거리 (마할라노비스/유클리디언 거리)

MYONGJI UNIVERSITY

특징의 분별력 | 분별력 측정

예제 9.1 부류간 거리

그림 9.4는 $X_i = \{(1,1)^T,\ (1,2)^T\}$이고 $X_j = \{(3,1)^T,\ (4,1)^T,\ (4,2)^T\}$인 상황을 보여 준다. $N_i = 2$와 $N_j = 3$이다. 두 부류 ω_i와 ω_j에 간의 거리를 (9.3)을 이용하여 계산해 보면 2.760이 됨을 알 수 있다.

그림 9.4 부류간 거리 측정을 위한 예

$$d_{ij} = \frac{1}{2*3}(dist((1,1),(3,1)) + dist((1,1),(4,1)) + dist((1,1),(4,2)) + dist((1,2),(3,1)) + dist((1,2),(4,1)) + dist((1,2),(4,2)))$$

$$= \frac{1}{6}(2 + 3 + \sqrt{10} + \sqrt{5} + \sqrt{10} + 3) = 2.760$$

특징의 분별력 | 분별력 측정

- 분류기 성능
  - 만약 분류기로 SVM을 사용하기로 결정 하였다면, 특징 벡터 x를 SVM으로 평가
  - 주어진 특징 벡터 x에 대해 훈련집합을 가지고 SVM을 훈련하고 검증 집합을 가지고 성능 측정
  - 성능을 특징 벡터의 분별력으로 판단
  - 장점: 사용하고자 하는 분류기에 딱 맞는 특징 벡터를 찾을 수 있음
  - 단점: 새로운 특징 벡터마다 분류기를 훈련이 필요 → 상당한 계산 시간 필요

특징 선택 문제의 이해

- 간단한 두 가지 예
  - 개와 고양이를 분류하는데 눈의 개수와 같은 특징은 쓸모 없다.
  - 그림 9.5: 두 특징은 매우 높은 중복성 → 하나만 있어도 된다.

$$\left.\begin{array}{l}\left.\begin{array}{l}00001110\\00010010\\00100000\\01000000\end{array}\right| u=7\\ \left.\begin{array}{l}10011000\\10100110\\01000011\\00111100\end{array}\right| d=14\end{array}\right| T=21$$

$$\mathbf{x}=(x_1,x_2)^{\mathrm{T}}=(u/T,d/T)^{\mathrm{T}}=(1/3,2/3)^{\mathrm{T}}$$

그림 9.5 숫자 인식의 예에서 특징의 중복성

- 실제 상황에서는 이런 직관을 사용하기 힘들다.
  - 효과적인 알고리즘이 필요하다.

특징 선택 문제의 이해

- 특징 선택 문제

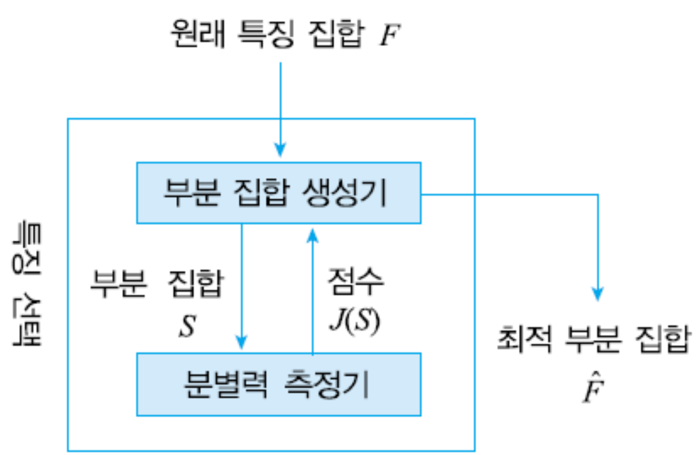


그림 9.6 특징 선택 알고리즘의 골격

  - 조합적 최적화 문제
    - 방대한 탐색 공간

표 9.1 d에 따른 특징 선택 시간의 추정

d	10	20	30	40	50
수행 시간	1초	17분	12일	35년	35700년
부분 집합 하나를 평가하는데 1/1000 초가 걸린다고 가정함					

특징 선택 문제의 이해

- 임의 탐색 알고리즘
  - 아무 생각 없이 여기 저기 뒤져보는 순진한 알고리즘

알고리즘 [9.1] **임의 탐색 알고리즘**

입력: 특징 집합 $F = \{x_1, x_2, \cdots, x_d\}$, 계산 시간 T, 부분 집합의 크기 $[\hat{d}_1, \hat{d}_2]$

출력: 부분 집합 $\hat{F}$

알고리즘:

1. $score = 0$;
2. 현재 시간 t를 0으로 설정한다.
3. **while** $(t < T)$ {
4. F의 부분 집합 S, $\hat{d}_1 \leq |S| \leq \hat{d}_2$ 를 임의로 생성한다.
5. $s = J(S)$; // (9.5)
6. **if** $(s > score)$ { $\hat{F} = S$; $score = s$;}
7. 현재 시간 t를 측정한다.
8. }
9. **return** $\hat{F}$;

특징 선택 문제의 이해

- 개별 특징 평가 알고리즘
  - 특징들 간의 상관 관계를 전혀 고려하지 않음
  - 특징의 중복성 (그림 9.5)을 무시

알고리즘 [9.2] 개별 특징 평가 알고리즘

입력: 특징 집합 $F=\{x_1, x_2, \cdots, x_d\}$, 부분 집합의 크기 $[\hat{d}_1, \hat{d}_2]$

출력: 부분 집합 $\hat{F}$

알고리즘:

1. **for** $(i=1$ **to** $d)$ $s_i = J(\{x_i\})$;
2. x_i를 s_i에 따라 내림차순으로 정렬한다.
3. $score = 0$;
4. **for** $(i=\hat{d}_1$ **to** $\hat{d}_2)$ {
5. 가장 좋은 i 개의 특징으로 부분 집합 S를 만든다.
6. $s = J(S)$;
7. **if** $(s > score)$ { $\hat{F} = S$; $score = s$;}
8. }
9. **return** $\hat{F}$;

탐색 알고리즘 | 전역 탐색 알고리즘

- 전역 탐색 알고리즘
  - 전체 특징 공간에서 가능성 있는 영역을 모두 탐색
  - 가능성 여부를 따지지 않고 모든 해를 평가하는 낱낱 탐색 알고리즘
  - 탐색 도중에 가능성이 없다고 판단되는 영역은 배제하여 효율을 높이는 한정 분기 알고리즘
  - 항상 전역 최적해를 보장
  - 하지만, 차원이 커지면 현실적인 시간 내에 해를 구하지 못하는 한계 존재

MYONGJI UNIVERSITY

탐색 알고리즘 | 전역 탐색 알고리즘

- 낱낱 탐색 알고리즘exhaustive search algorithm
  - 모든 해를 다 살피는 매우 우직한 알고리즘 (d가 큰 경우 비현실적)

알고리즘 [9.3] 낱낱 탐색 알고리즘

입력: 특징 집합 $F = \{x_1, x_2, \cdots, x_d\}$, 부분 집합의 크기 $[\hat{d}_1, \hat{d}_2]$

출력: 부분 집합 $\hat{F}$

알고리즘:

$score$=0;
while (TRUE) {
    S = next_subset(F, $\hat{d}_1, \hat{d}_2$); // 다음 부분 집합을 생성한다.
    if ($S \neq$NULL) {
        s = $J(S)$; // (9.5)
        if (s>$score$) { $\hat{F} = S$; $score$=s;}
    }
    else **break**;
}
return $\hat{F}$;

탐색 알고리즘 | 전역 탐색 알고리즘

- 한정 분기 알고리즘branch and bound algorithm
  - 탐색 과정에서 가능성 없는 영역을 배제하여 계산 효율 얻음
  - 단조성 조건을 만족하는 경우에 적용 가능

알고리즘 [9.4] 한정 분기 알고리즘

입력: 특징 집합 $F = \{x_1, x_2, \cdots, x_d\}$, 부분 집합의 크기 $[\hat{d}_1, \hat{d}_2]$

출력: 부분 집합 $\hat{F}$

알고리즘:

// $|S|$ = 집합 S의 크기

// next_children(S): S보다 크기가 하나 줄어든 부분 집합을 생성하는 함수

$score = 0$;

BranchBound(F); // 순환 호출을 기동시킴

return $\hat{F}$;

BranchBound(S) {

  for (next_children(S)로 생성되는 부분 집합 S_1 각각에 대해) {

    $s = J(S_1)$;

    if ($s > score$) {

      if ($|S_1| = \hat{d}_2$) { $\hat{F} = S_1$; $score = s$;}

      else if ($|S_1| > \hat{d}_2$) BranchBound(S_1); // 순환 호출

    }

  }

}

MYONGJI UNIVERSITY

탐색 알고리즘 | 전역 탐색 알고리즘

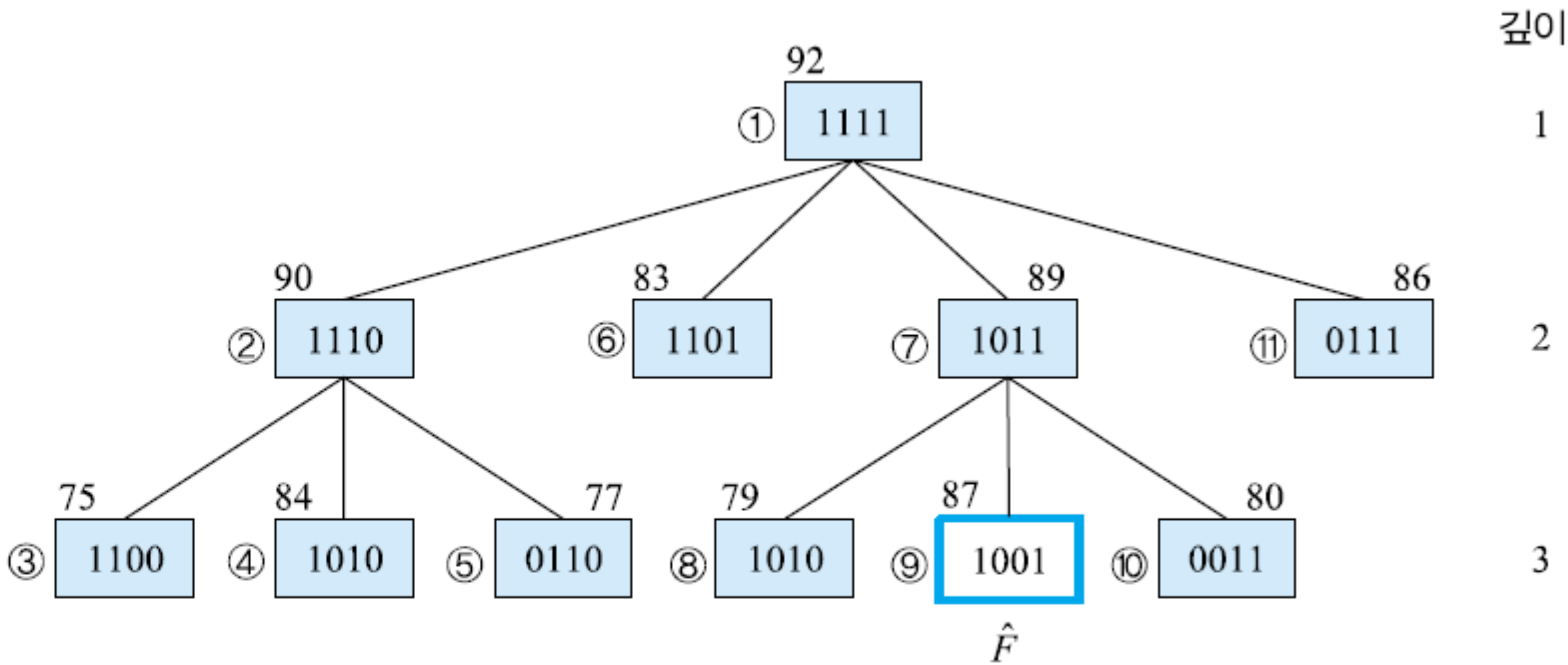


그림 9.7 한정 분기의 예 ($d=4$와 $[\hat{d}_1, \hat{d}_2]=[2,2]$인 경우)

MYONGJI UNIVERSITY

탐색 알고리즘 | 순차 탐색 알고리즘

두 가지 부분 집합을 정의해 보자. 알고리즘이 수행되는 어느 순간에 F_1은 선택된 특징의 집합을 지칭하며 F_2는 F_1의 여집합이다. 즉 $F = F_1 \cup F_2$이다. 이제 지역 탐색 연산 두 가지를 정의해 보자. (9.7)의 연산 *add*는 F_1에 특징 하나를 추가한다. 추가할 특징은 F_2에서 고르는데, 추가했을 때 가장 큰 성능 증가를 가져오는 특징 x_k가 F_2에서 F_1로 이동한다. (9.8)의 연산 *rem*은 F_1에서 특징 하나를 제거한다. 제거했을 때 가장 적은 성능 저하를 가져오는 특징 x_k가 F_1에서 F_2로 이동한다.

$$add: \quad x_k = \underset{x_i \in F_2}{\arg\max}\, J(F_1 \cup \{x_i\}) \text{ 를 } F_2\text{에서 } F_1\text{로 옮겨라.} \tag{9.7}$$

$$rem: \quad x_k = \underset{x_i \in F_1}{\arg\max}\, J(F_1 - \{x_i\}) \text{ 를 } F_1\text{에서 } F_2\text{로 옮겨라.} \tag{9.8}$$

MYONGJI UNIVERSITY

탐색 알고리즘 | 순차 탐색 알고리즘

- SFS 알고리즘sequential forward search algorithm
  - 한번에 하나씩 특징을 추가해 나가는 방식

알고리즘 [9.5] SFS 알고리즘

입력: 특징 집합 $F = \{x_1, x_2, \cdots, x_d\}$, 부분 집합의 크기 $[\hat{d}_1, \hat{d}_2]$

출력: 부분 집합 $\hat{F}$

알고리즘:

$score$=0;
$F_1=\varnothing$; $F_2=F$; // F_1은 공집합, F_2는 F를 가지고 출발
for $(i=1$ **to** $\hat{d}_2)$ {
    add;
    if $(\hat{d}_1 \le |F_1| \le \hat{d}_2)$ {
        $s = J(F_1)$;
        if $(s > score)$ $\{\hat{F} = F_1;\ score = s;\}$
    }
}
return $\hat{F}$;

MYONGJI UNIVERSITY

탐색 알고리즘 | 순차 탐색 알고리즘

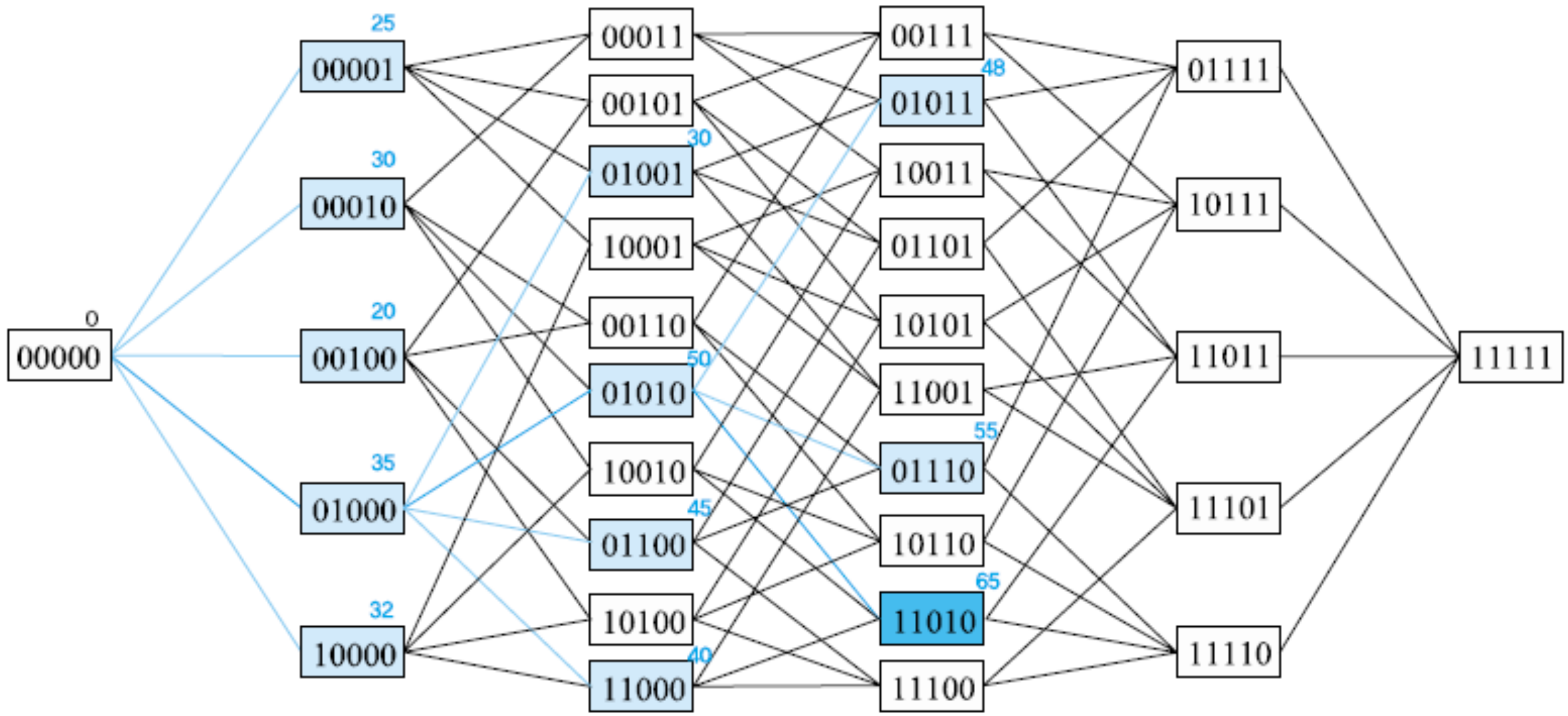


그림 9.8 SFS 알고리즘의 동작 ($d=5$와 $[\hat{d}_1, \hat{d}_2]=[2,3]$인 경우)

탐색 알고리즘 | 순차 탐색 알고리즘

- PTA 알고리즘plus-*p*-take-away-*q* algorithm
  - *p* 개의 특징을 추가하고 *q* 개를 제거하는 연산을 반복

알고리즘 [9.7] PTA (plus-*p*-take-away-*q*) 알고리즘

입력: 특징 집합 $F = \{x_1, x_2, \cdots, x_d\}$, p와 q $(p > q)$, 부분 집합의 크기 $[\hat{d}_1, \hat{d}_2]$

출력: 부분 집합 $\hat{F}$

알고리즘:

```
score = 0;
F1 = ∅; F2 = F;   // F1은 공집합, F2는 F를 가지고 출발
while (|F1| ≤ d̂2) {
    for (i = 1 to p) add;
    for (i = 1 to q) rem;
    if (d̂1 ≤ |F1| ≤ d̂2) {
        s = J(F1);
        if (s > score) {F̂ = F1; score = s;}
    }
}
return F̂;
```

탐색 알고리즘 I
통계적 탐색 연산을 가진 알고리즘

- 모든 순차 탐색 알고리즘은 욕심greedy 알고리즘이다.
  - 전역 최적점이 아니라 지역 최적점에 빠질 가능성
- 이를 극복하기 위한 통계적 탐색 알고리즘
  - 시뮬레이티드 어닐링 (담금질 기법)
  - 유전 알고리즘

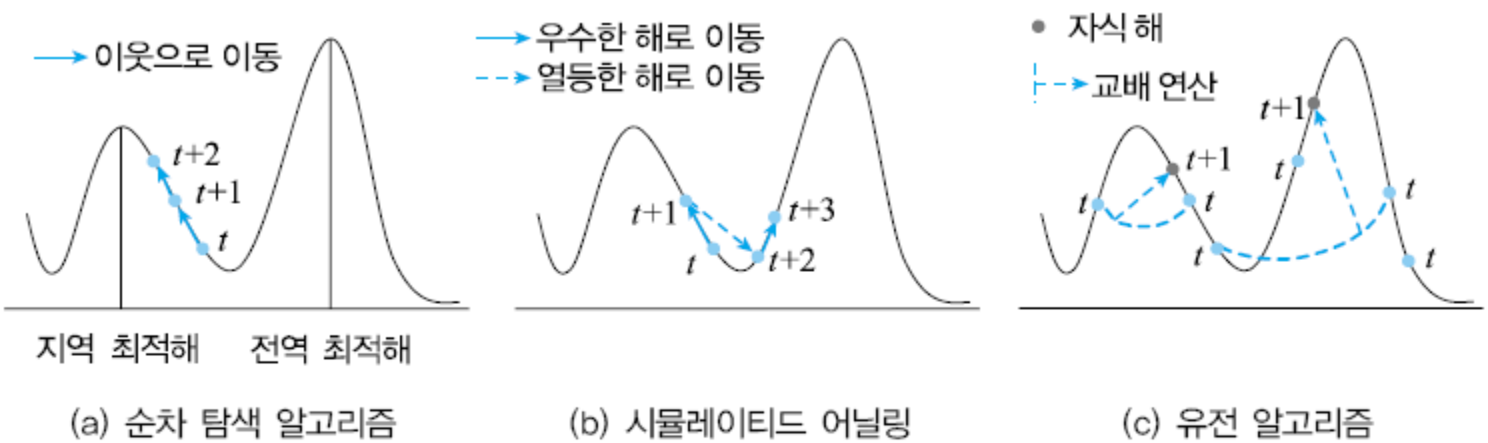


그림 9.10 여러 탐색 알고리즘의 동작 원리 (최대화 문제)

Thank you

MYONGJI
UNIVERSITY