9장. 특징 선택

오일석, 패턴인식, 교보문고, 2008.

들어가는 말

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

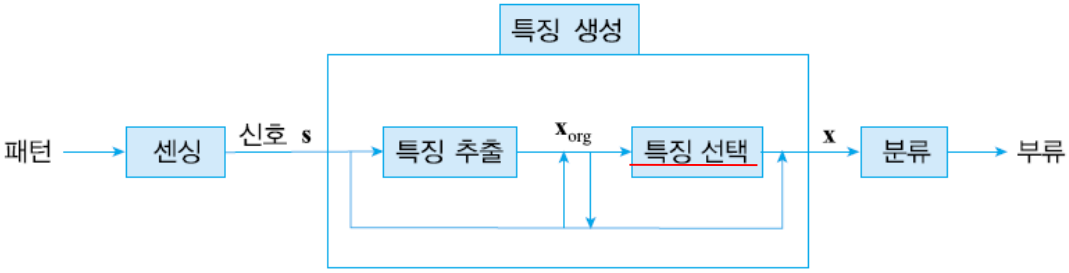
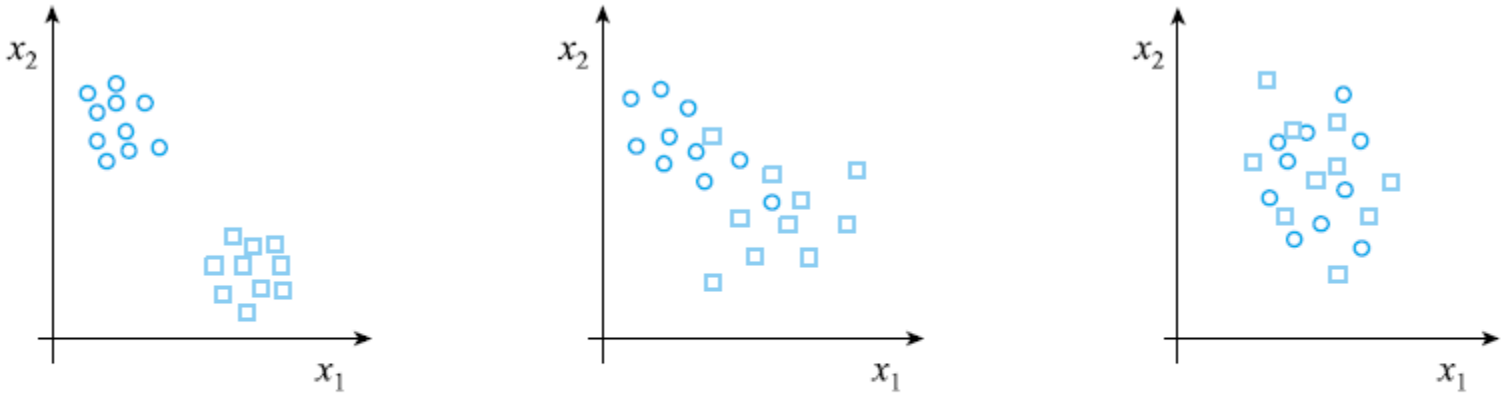


그림 8.2 특징 생성의 절차

9.1 특징의 분별력
9.1.1 직관적 이해



(a) 부류내 분산은 작고 부류간 분산은 큰 상황

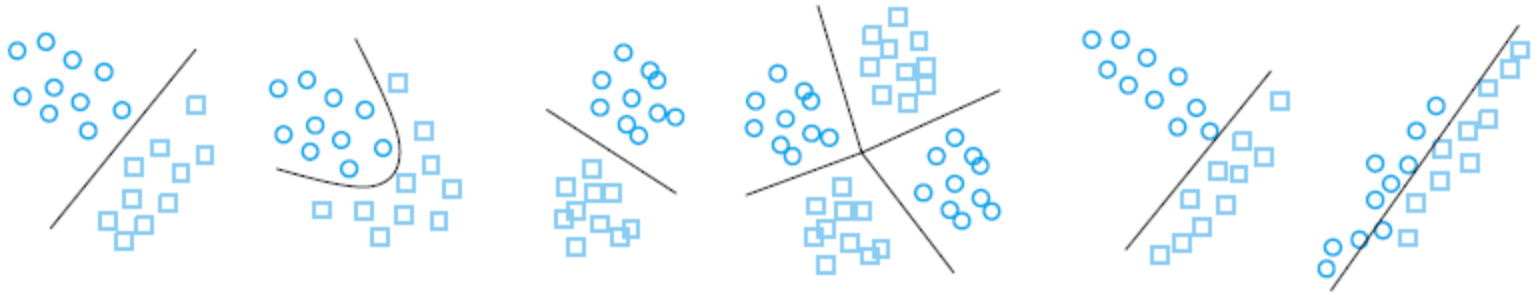
(b) 부류내 분산은 크고 부류간 분산은 작은 상황

(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

9.1.1 직관적 이해



(a) 선형 분리 가능과 불가능　　(b) 단일 모드와 다중 모드　　(c) 다른 공분산과 같은 공분산

그림 9.2　샘플 분포가 만드는 여러 가지 모양

지금까지는 2 차원 특징 공간을 시각적으로 표현하고 거기에서 직관적인 이해를 시도하였다. 하지만 실제 세계는 매우 높은 차원의 데이터를 다루므로 이러한 시도가 불가능하다. 예를 들어 8.5.3절의 얼굴 인식에서 고유 얼굴 특징 추출은 수십 차원의 특징 벡터를 만들어 준다. 필기 문자 인식에서는 수십~수백 차원의 특징 벡터를 사용하는 것이 보통이다. 이런 상황에서는 결국 수학의 힘을 빌릴 수밖에 없다.

9.1.2 분별력 측정

앞 절에서 2 차원 특징 공간의 그림을 통해 특징 벡터 **x**의 성질을 직관적으로 살펴보았다. 이제는 **x**의 좋고 나쁨을 수량적으로 평가할 수 있는 기준 함수에 대해 공부하자. 특징 벡터 **x**가 얼마나 좋은 지의 척도를 **분별력**이라 discriminatory power 부른다. 또는 **부류 분리** class separation 능력이라 부른다. 곧이어 널리 사용되는 척도 세 가지를 제시한다.

- 다이버전스
- 훈련 샘플의 거리
- 분류기 성능

9.1.2 분별력 측정

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

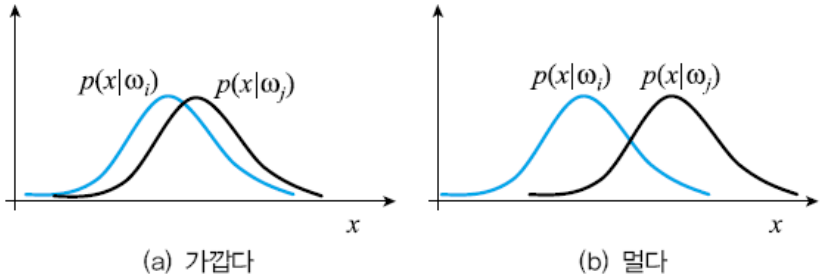


그림 9.3 두 확률 분포 간의 거리

  - 거리 측정은 어떻게 하나? KL 다이버전스 활용 (부록 A)

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

2 부류

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

M 부류

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

9.1.2 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

9.1.2 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

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

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

9.1.2 분별력 측정

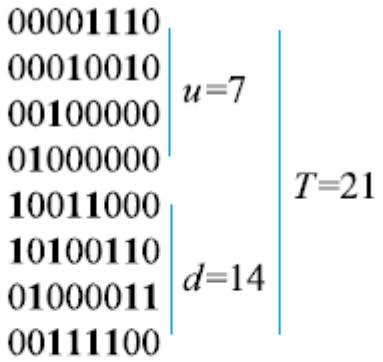
- 분류기 성능

더욱 현실적인 척도도 있다. 예를 들어 분류기로 SVM을 사용하기로 결정하였다면 특징 벡터 **x**를 SVM으로 평가할 수 있다. 주어진 특징 벡터 **x**에 대하여 훈련 집합을 가지고 SVM을 훈련하고 검증 집합을 가지고 성능을 측정한다. 이렇게 얻은 성능을 특징 벡터 **x**의 분별력으로 취한다.

앞의 다른 방법에 견주어 이 방법의 장점은 사용하고자 하는 분류기에 딱 맞는 특징 벡터를 찾아낼 수 있다는 점이다. 예를 들어 다이버전스를 사용한다면 최적의 특징 벡터를 결정하였다 하더라도 그것이 SVM에서도 최적이라는 보장은 없는 것이다. 단점은 새로운 특징 벡터마다 분류기를 훈련해야 하는데 이때 드는 계산 시간이 과다하다는 점이다. 최적의 특징 벡터를 찾기 위해서는 아주 많은 수의 서로 다른 특징 벡터를 평가해야 하기 때문이다. 평가해야 하는 특징 벡터의 수에 대해서는 곧바로 이어지는 9.2절에서 공부하기로 하자.

9.2 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.



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

그림 9.5 숫자 인식의 예에서 특징의 중복성

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

9.2 특징 선택 문제의 이해

- 특징 선택 문제

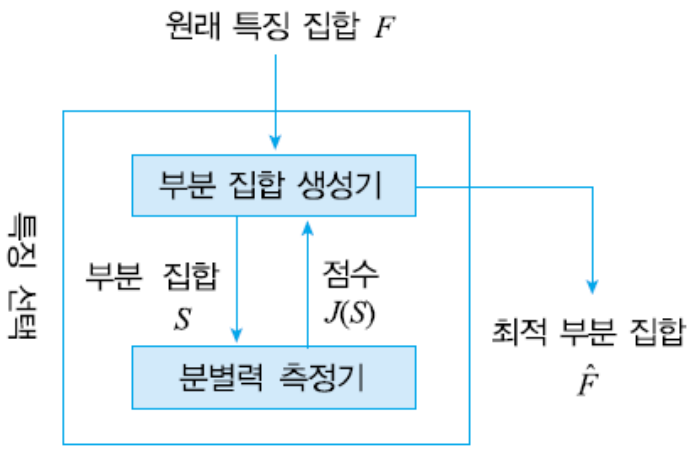


그림 9.6 특징 선택 알고리즘의 골격

  - 조합적 최적화 문제
    - $\Theta(2^d)$의 계산 복잡도. 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					

9.2 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 $S, \hat{d}_1 \le |S| \le \hat{d}_2$ 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** $\hat{F}$;

9.2 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** $\hat{F}$;

9.3 전역 탐색 알고리즘

전역 탐색 global search 알고리즘은 전체 특징 공간에서 가능성 있는 영역을 모두 탐색하는 알고리즘이다. 가능성 여부를 따지지 않고 모든 해를 평가하는 낱낱 탐색 알고리즘과 탐색 도중에 가능성이 없다고 판단되는 영역은 배제하여 효율을 높이는 한정 분기 알고리즘이 있다. 이 둘은 탐색 공간 전체를 대상으로 하므로 항상 전역 최적 해를 보장한다. 하지만 앞에서 언급한 바와 같이 d가 커지면 현실적인 시간 내에 해를 구하지 못하는 한계가 있다. 한정 분기는 낱낱 탐색에 비해 효율적이지만 이 한계를 극복하지는 못한다.

9.3 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:

```
1.  score=0;
2.  while (TRUE) {
3.      S = next_subset(F, d̂1, d̂2);    // 다음 부분 집합을 생성한다.
4.      if (S≠NULL) {
5.          s = J(S);    // (9.5)
6.          if (s>score) { F̂ = S ; score=s;}
7.      }
8.      else break;
9.  }
10. return F̂ ;
```

9.3 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
1.  score = 0;
2.  BranchBound(F);   // 순환 호출을 기동시킴
3.  return F̂;

4.  BranchBound(S) {
5.      for (next_children(S)로 생성되는 부분 집합 S1 각각에 대해) {
6.          s = J(S1);
7.          if (s > score) {
8.              if (|S1| = d̂2) { F̂ = S1; score = s;}
9.              else if (|S1| > d̂2) BranchBound(S1);   // 순환 호출
10.         }
11.     }
12. }
```

9.3 전역 탐색 알고리즘

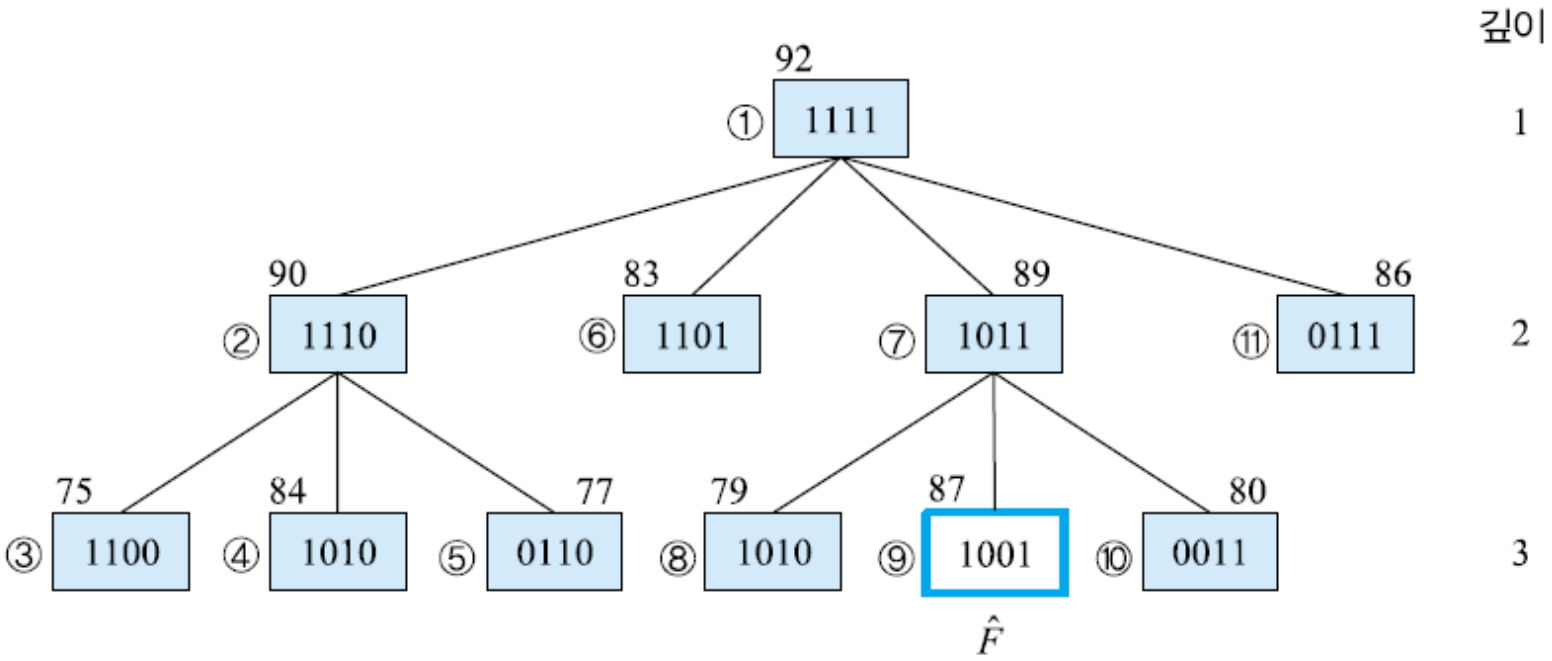


그림 9.7 한정 분기의 예 ($d=4$와 $[\hat{d}_1, \hat{d}_2]=[2,2]$인 경우)

9.4 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.

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

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

9.4 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:

```
1. score=0;
2. F1=∅; F2=F;   // F1은 공집합, F2는 F를 가지고 출발
3. for (i = 1 to d̂2) {
4.       add;
5.       if (d̂1 ≤| F1 |≤ d̂2) {
6.            s = J(F1);
7.            if (s>score) { F̂ = F1; score=s;}
8.       }
9. }
10. return F̂;
```

9.4 순차 탐색 알고리즘

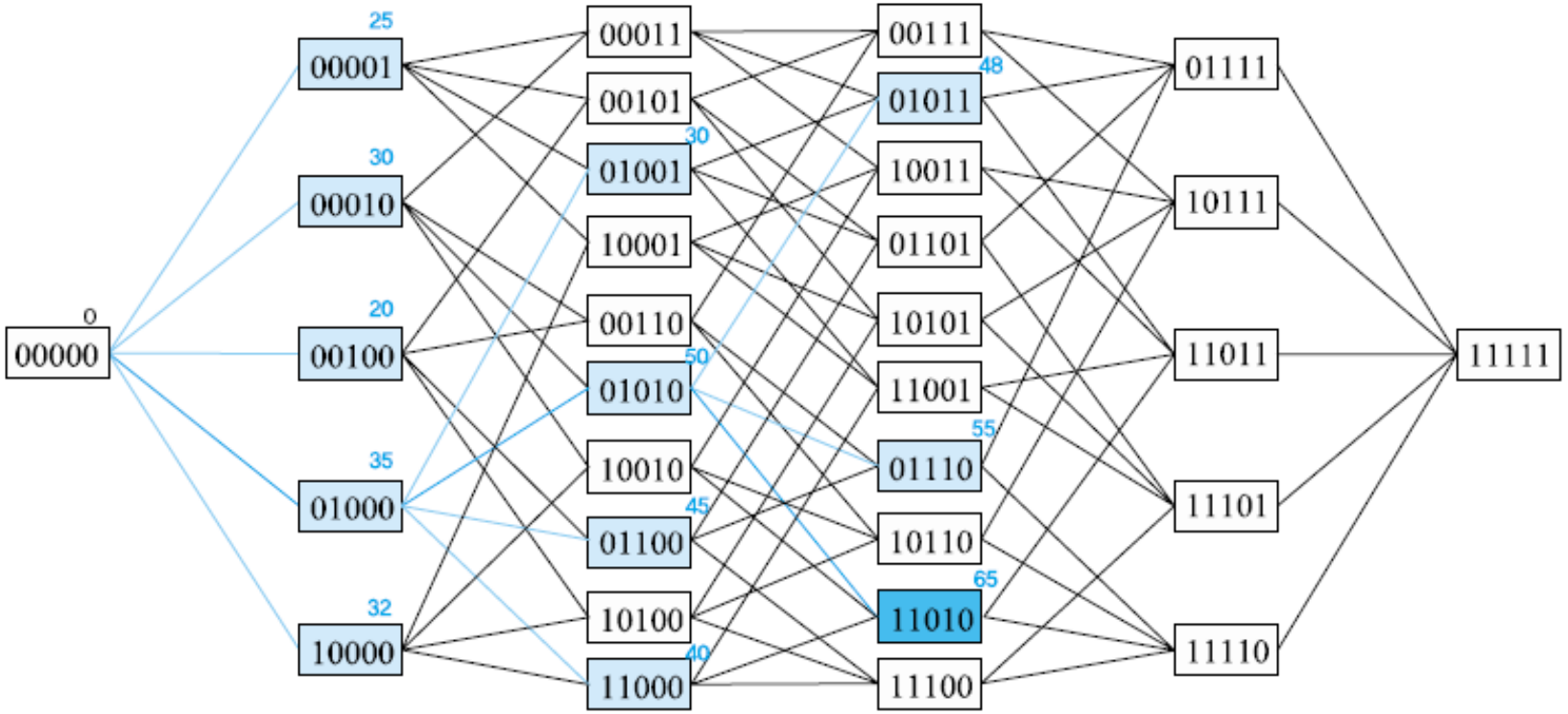


그림 9.8 SFS 알고리즘의 동작 ($d=5$와 $[\hat{d}_1, \hat{d}_2]=[2,3]$인 경우)

9.4 순차 탐색 알고리즘

- PTA 알고리즘plus-*p*-take-away-*q* algorithm
  - *p* 개의 특징을 추가하고 *r* 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:

```
1.  score = 0;
2.  F1 = ∅; F2 = F;   // F1은 공집합, F2는 F를 가지고 출발
3.  while (|F1| ≤ d̂2) {
4.      for (i = 1 to p) add;
5.      for (i = 1 to q) rem;
6.      if (d̂1 ≤ |F1| ≤ d̂2) {
7.          s = J(F1);
8.          if (s > score) {F̂ = F1; score = s;}
9.      }
10. }
11. return F̂;
```

9.5 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
  - 유전 알고리즘
  - 11장에서 자세히 다룸

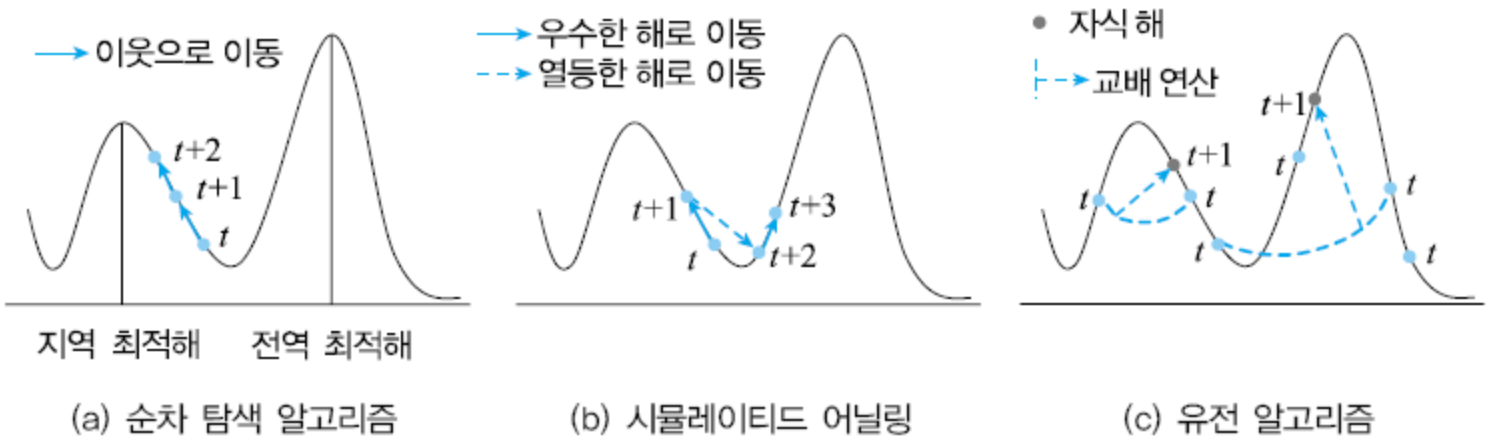


그림 9.10 여러 탐색 알고리즘의 동작 원리 (최대화 문제)