

9. 특징 선택

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

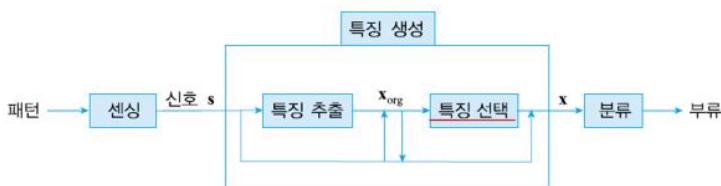


그림 8.2 특징 생성의 절차

9.1.1 직관적 이해

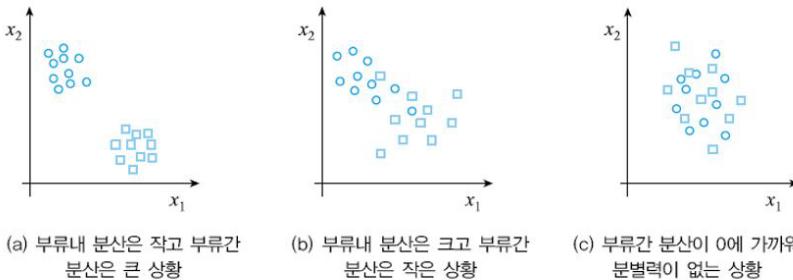
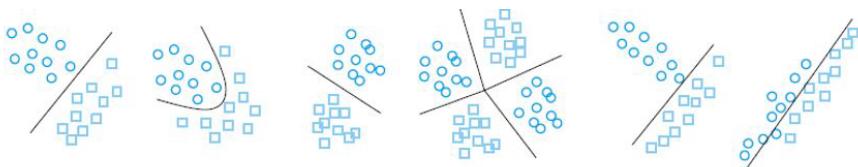


그림 9.1 특징 벡터 $x = (x_1, x_2)^T$ 의 분별력

• 부류내 분산과 부류간 분산

- **within class variance**: 같은 부류에 속하는 샘플이 얼마나 퍼져있는지 척도
- **between class variance**: 서로 다른 부류의 분포가 얼마나 떨어져 있는지 척도
- 부류내 분산은 작고, 부류간 분산은 크게 해줄수록 좋은 특징 벡터



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

그림 9.2 샘플 분포가 만드는 여러 가지 모양

실제로는 2차원이 아닌 수십 ~ 수백 차원의 특징 벡터를 사용하므로,
수학의 힘을 발휘야 함.

9.1.2 분별력 측정

특징 벡터 x 가 얼마나 좋은지의 척도: 분별력, 뷔류분리

① 다이버전스

- 확률 분포 간의 거리를 이용 (거리를 멀게 해주는 특징은 수록 좋다)

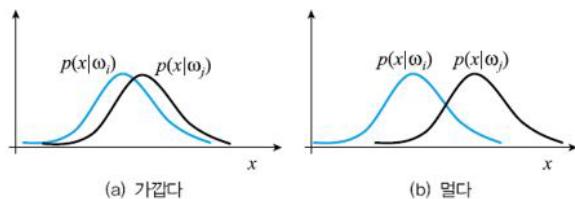


그림 9.3 두 확률 분포 간의 거리

$$* \text{KL}(P_1(x), P_2(x)) = \sum_x P_1(x) \log_2 \frac{P_1(x)}{P_2(x)}$$

- 거리 측정은? KL divergence 활용

$$d_{ij} = \text{KL}(P(X|w_i), P(X|w_j)) + \text{KL}(P(X|w_i), P(X|w_j))$$

$$d = \sum_{i=1}^M \sum_{j=1}^M P(w_i) P(w_j) d_{ij}$$

- 현실적 문제: 확률 분포를 알 때 적용 가능

→ 확률 추정이 안고 있는 차원의 저주 같은 문제를 이어받는다.

② 훈련 샘플의 거리

- 훈련 집합에 있는 샘플을 가지고 직접 측정(현실 적용이 쉽다)

$$d_{ij} = \frac{1}{N_i N_j} \sum_{k=1}^{N_i} \sum_{m=1}^{N_j} \text{dist}(x_i^k, x_j^m)$$

부류 w_j 의 m 번째 샘플
두 점 x_i 와 x_j 간의 거리
(마할라노비스 or 유clidean 거리를 많이 사용)

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_i 와 ω_j 에 간의 거리를 (9.3)을 이용하여 계산해 보면 2.760이 됨을 알 수 있다.

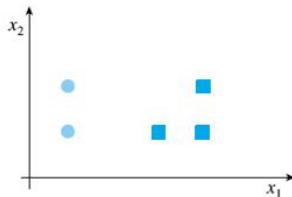


그림 9.4 부류간 거리 측정을 위한 예

$$\begin{aligned} d_{ij} &= \frac{1}{2 \times 3} (\text{dist}((1,1), (3,1)) + \text{dist}((1,1), (4,1)) + \text{dist}((1,1), (4,2)) \\ &\quad + \text{dist}((1,2), (3,1)) + \text{dist}((1,2), (4,1)) + \text{dist}((1,2), (4,2))) \\ &= \frac{1}{6} (2 + 3 + \sqrt{10} + \sqrt{5} + \sqrt{10} + 3) = 2.760 \end{aligned}$$

③ 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 훈련집합을 가지고 SVM을 훈련하고 검증 집합을 가지고 성능을 측정한다. 이렇게 얻은 성능을 특징 벡터 x 의 분별력으로 취한다.

장점: 분류기에 딱 맞는 특징 벡터를 찾아낼 수 있다.

ex) 다이버전스를 사용하면 최적의 특징 벡터를 결정하였다 하더라도 그것이 SVM에서도 최적이라는 보장은 없다.

단점: 새로운 특징 벡터마다 분류기를 훈련해야 하는데 이때 드는 계산 시간이 과하다.

9.2 특징 선택 문제의 이해

- 간단한 두 가지 예
- 개와 고양이를 분류하는데 눈의 개수와 같은 특징은 쓸모없다.

00001110	
00010010	$u=7$
00100000	
01000000	
10011000	$T=21$
10100110	
01000011	$d=14$
00111100	

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

그림 9.5 숫자 인식의 예에서 특징의 중복성

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

- 특징 선택 문제

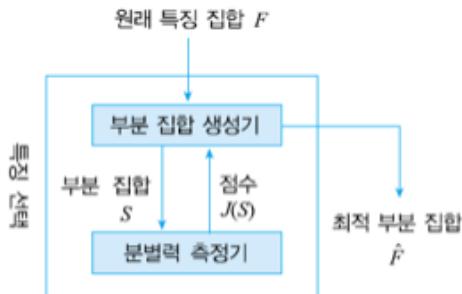


그림 9.6 특징 선택 알고리즘의 골격

- 조합적 최적화 문제

- $\Theta(2^d)$ 의 계산 복잡도. 방대한 탐색 공간

표 9.1 d 에 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					

알고리즘 [9.1] 임의 탐색 알고리즘

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

출력: 부분 집합 \hat{F}

알고리즘:

1. $score = 0;$
2. 현재 시간 t 를 0으로 설정한다.
3. **while** ($t < T$) {
4. F 의 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 t 를 측정한다.
8. }
9. **return** $\hat{F};$

→ 매우 생각없이
여기저기 뒤져보는
매우 비효율적인
알고리즘

입력: 특징 집합 $F = \{x_1, x_2, \dots, x_d\}$, 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_i 를 s_i 에 따라 내림차순으로 정렬한다.
3. $score = 0$;
4. **for** ($i = \hat{d}_1$ **to** \hat{d}_2) {
 5. 가장 좋은 i 개의 특징으로 부분 집합 S 를 만든다.
 6. $s = J(S)$;
 7. **if** ($s > score$) { $\hat{F} = S$; $score = s$; }
 8. }
9. **return** \hat{F} ;

특징을 개별적으로 평가, 특징들 간의 상관관계를
아예 무시. 그럼 9.5에서 둘다 선택되거나
둘다 선택되지 않는 비효율적인 결과가
나옴.

9.3 전역 탐색 알고리즘

알고리즘 전체 특징 공간에서 가능성 있는 영역을 모두 탐색하는 알고리즘

탐색 공간 전체를 대상으로 하므로 항상 전역 최적해 보장

알고리즘 [9.3] 낱낱 탐색 알고리즘 : 모든 해를 평가

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

출력: 부분 집합 \hat{F}

알고리즘:

1. $score=0$;
2. **while** (TRUE) {
 - 시간 한계 X
 3. $S = next_subset(F, \hat{d}_1, \hat{d}_2)$; // 다음 부분 집합을 생성한다.
 4. **if** ($S \neq \text{NULL}$) {
 5. $s = J(S)$; // (9.5)
 6. **if** ($s > score$) { $\hat{F} = S$; $score = s$; }
 7. }
 8. **else break**;
 9. }
 10. **return** \hat{F} ;

알고리즘 [9.4]

한정 분기 알고리즘

가능성이 없다고 판단되는 영역을 배제하여

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

출력: 부분 집합 \hat{F}

알고리즘:

// $|S| =$ 집합 S 의 크기

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

1. $score = 0$;
2. $BranchBound(F)$; // 순환 호출을 기동시킴
3. **return** \hat{F} ;

4. $BranchBound(S) \{$
5. **for** (next_children(S)로 생성되는 부분 집합 S_1 각각에 대해) {
6. $s = J(S_1)$;
7. **if** ($s > score$) {
8. **if** ($|S_1| = \hat{d}_2$) { $\hat{F} = S_1$; $score = s$; }
9. **else if** ($|S_1| > \hat{d}_2$) **BranchBound(S_1)**; // 순환 호출
10. }
11. }
12. }

* 단, 단조성(monotonicity)

조건이 만족되어야 함

$S_1 | S_2$ 이면

$$J(S_1) \leq J(S_2)$$

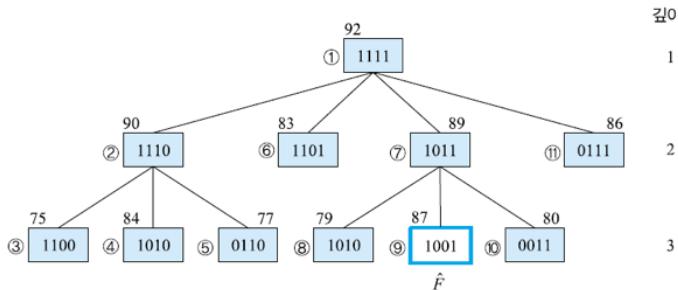


그림 9.7 한정 분기의 예 ($d=4$ 와 $[\hat{d}_1, \hat{d}_2] = [2,2]$ 인 경우)

9.4 순차 선택 알고리즘

F_1 은 선택된 특징의 집합이며, F_2 는 F_1 의 여집합이다. (즉, $F = F_1 \cup F_2$)

이제) 지역 탐색 연산 두 가지를 정의해 보자.

add: $x_k = \underset{x_i \in F_2}{\operatorname{argmax}} J(F_1 \cup \{x_i\})$ 를 F_2 에서 F_1 로 옮겨라.

rem: $x_k = \underset{x_i \in F_1}{\operatorname{argmax}} J(F_1 - \{x_i\})$ 를 F_1 에서 F_2 로 옮겨라

1. SFS

반대↓

알고리즘 [9.5]

SFS 알고리즘 (SBS 는 $F_1 = F$ 를 가지고 출발)

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

출력: 부분 집합 \hat{F}

알고리즘:

1. $score=0;$
2. $F_1=\emptyset; F_2=F; // F_1$ 은 공집합, F_2 는 F 를 가지고 출발
3. **for** ($i=1$ **to** \hat{d}_2) {
4. **add**;
5. **if** ($\hat{d}_1 \leq |F_1| \leq \hat{d}_2$) {
6. $s = J(F_1);$
7. **if** ($s > score$) { $\hat{F} = F_1; score=s;$ }
8. }
9. }
10. **return** $\hat{F};$

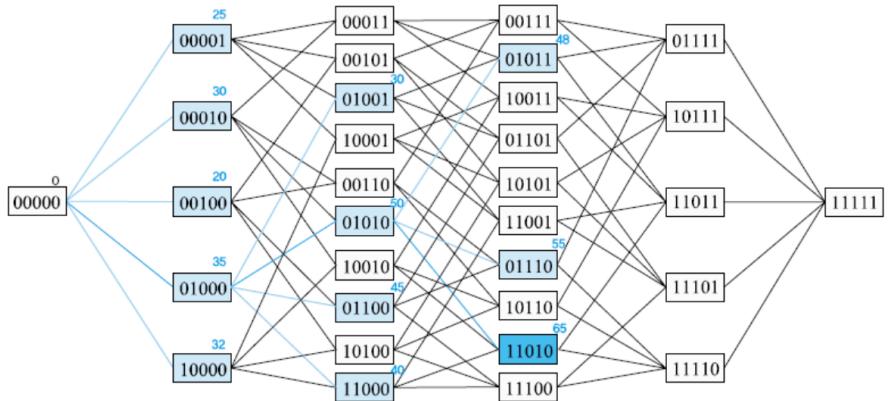


그림 9.8 SFS 알고리즘의 동작 ($d=5$ 와 $\hat{d}_1, \hat{d}_2 = [2,3]$ 인 경우)

but, 한번 추가한 특징은 다시 빼낼 수 없고 제거한건 다시 넣을 수 X

★ PTA 알고리즘

- p 개의 특징을 추가하고 r 개를 제거하는 연산을 반복

알고리즘 [9.7]

PTA (plus-p-take-away-q) 알고리즘

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

출력: 부분 집합 \hat{F}

알고리즘:

1. $score = 0;$
2. $F_1 = \emptyset; F_2 = F;$ // F_1 은 공집합, F_2 는 F 를 가지고 출발
3. **while** ($|F_1| \leq \hat{d}_2$) {
4. **for** ($i = 1$ to p) *add*;
5. **for** ($i = 1$ to q) *rem*;
6. **if** ($\hat{d}_1 \leq |F_1| \leq \hat{d}_2$) {
7. $s = J(F_1);$
8. **if** ($s > score$) { $\hat{F} = F_1$; $score = s$; }
9. }
10. }
11. **return** $\hat{F};$

→ 그러나 정해 놓은 크기만큼씩
추가, 삭제

→ 상황에 맞게 적응적으로?
SFS 알고리즘

9.5 통계적 탐색 연산을 가진 알고리즘

- 모든 순차 탐색 알고리즘은 greedy 알고리즘이다.
→ 전역 최저점이 아니라 지역 최저점에 빠질 가능성
- 이를 극복하기 위한 통계적 탐색 알고리즘
 - simulated annealing)→ 11장에서 자세히
 - genetic algorithm

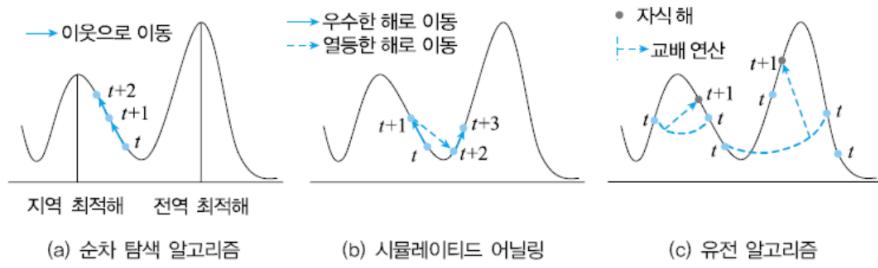


그림 9.10 여덟 탐색 알고리즘의 동작 원리 (최대화 문제)