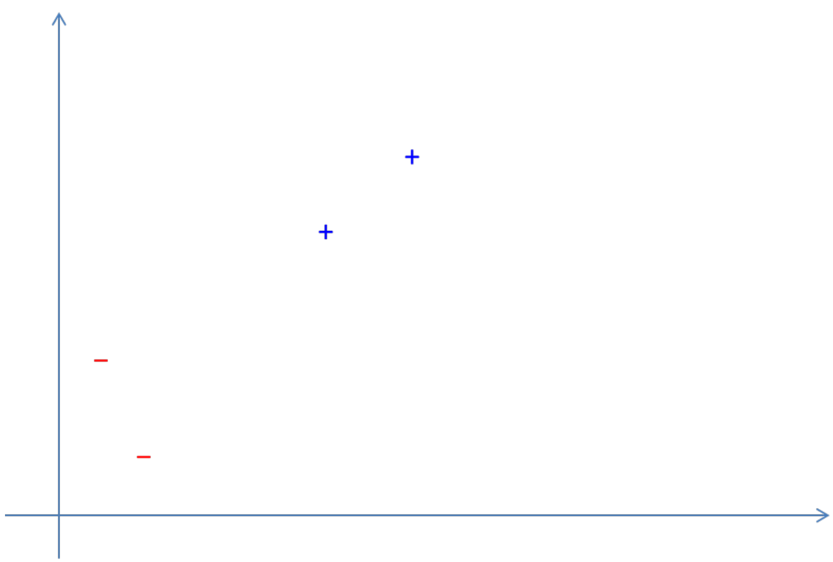
SVM (Support vector machines)

Sang Yup Lee

SVM

- Introduction to SVM
  - 분류의 문제에 주로 적용
  - 관측치들을 벡터로 변환하여 공간상의 점으로 표현



두개는 +에 해당하는 레이블 (즉, 종속변수 값)을 그리고 나머지 두개는 –에 해당하는 레이블 정보, 보통 $y_i \in \{-1, 1\}$ 라고 가정

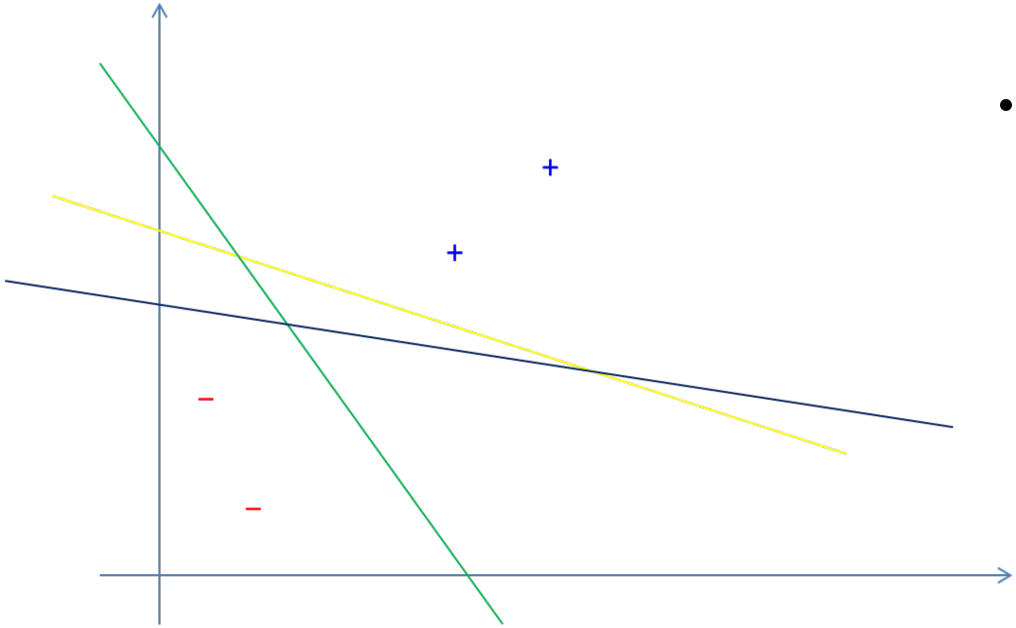
감성분석에서 +는 긍정, –는 부정을 의미한다고 생각 가능

SVM

- Intro to SVM (cont'd)
  - 레이블에 따라 벡터들을 구분하기 위해서 하이퍼플레인 (hyperplane) 사용
  - n 차원 공간에 대한 하이퍼플레인
    - $\mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 + \cdots + \mathrm{b}_\mathrm{n} x_\mathrm{n} = 0$
    - 예) 2차원 공간인 경우: $\mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = 0$

SVM

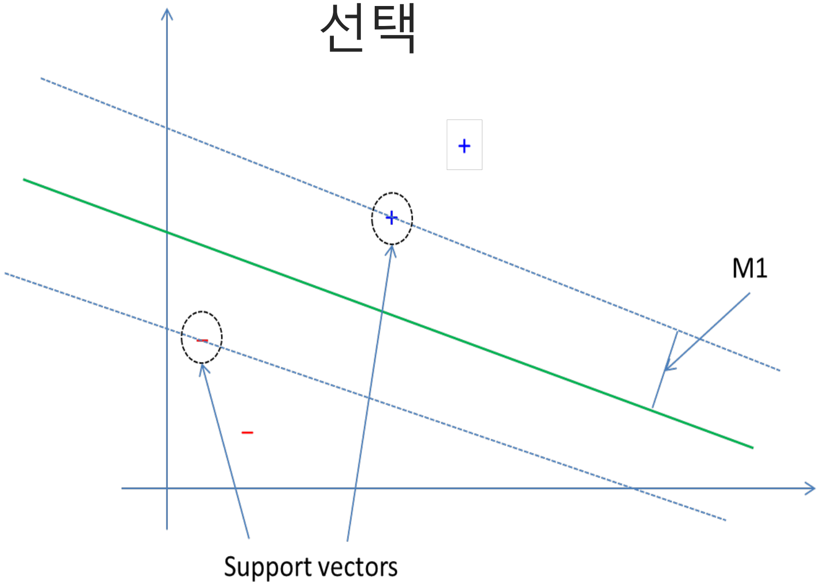
- 2차원 공간의 경우
  - 레이블에 따라 관측치들을 구분하는 여러 개의 직선 존재

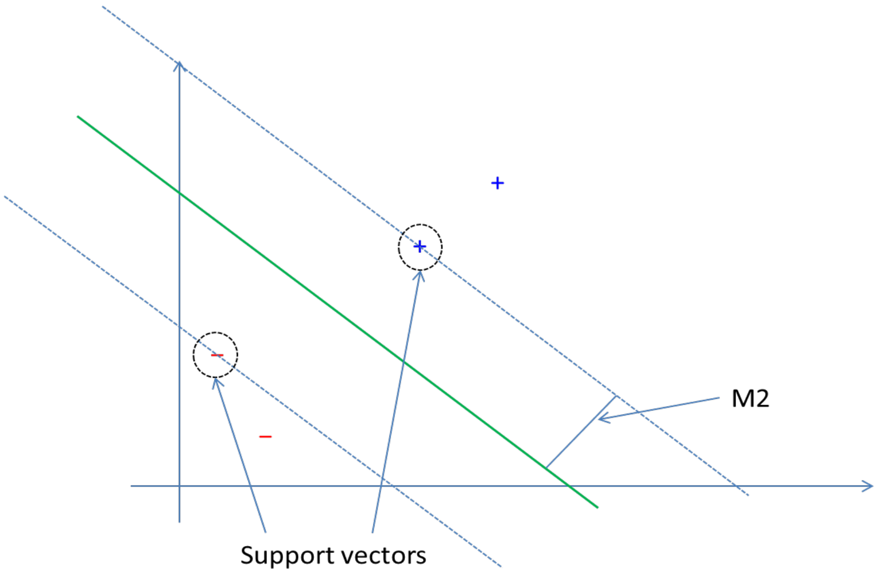


- 여러개의 직선 존재 ⇒
  - 어떠한 직선을 사용해야 하는가?
  - 어떠한 직선이 최적인가?

SVM

- Intro to SVM (cont'd)
  - 하이퍼플레인과 가장 가까이 있는 벡터들(서포트 벡터라고 함)과의 거리(margin이라고 함)가 가장 크게 하는 하이퍼플레인 선택





If M2 > M1, then the hyperplane on the right side is preferred.

SVM

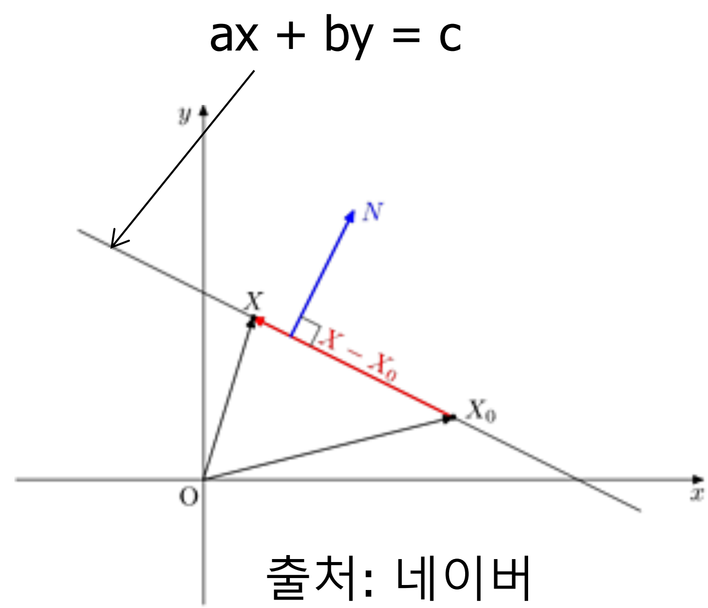
- Intro to SVM (cont'd)
  - 2차원에 대한 하이퍼플레인: 직선 형태
    - $\mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = 0$
  - 우리는 margin을 가장 크게하는 $\mathrm{b}_0, \mathrm{b}_1, \mathrm{b}_2$을 찾아야 하는 것
    - $\underset{\boldsymbol{b}}{\operatorname{argmax}}\, M$
      - M = Margin
  - Margin은 어떻게 표현이 되는가?
    - 이를 위해서는 점과 직선 사이의 거리를 알아야 한다.

SVM

- 점과 직선 사이의 거리
  - 점(x_0, y_0)과 직선 ax + by + c = 0 사이의 거리
    - ${|ax_0+by_0+c|}/{\sqrt{a^2+b^2}}$
    - 여기에서 $ax_0 + by_0 + c = \mathrm{k}$라고 표현 가능, 여기서 점$(x_0, y_0)$은 직선 $ax + by + c = \mathrm{k}$ 위의 점이 되는 것이고, 이 직선 $(ax + by + c = \mathrm{k})$은 $ax + by + c = 0$으로 표현되는 hyperplane과 평행한 직선

참고

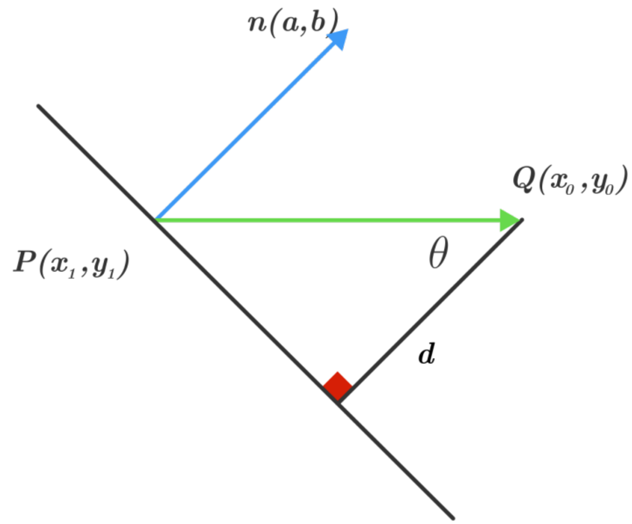
- 법선 벡터 (normal vector)
  - 직선: $ax + by = c$
    의 법선벡터는 $N = (a, b)$
  - 내적을 이용해 계산
  - 해당 직선을 지나는 임의의
    두 점 ($X = (x,y)$, $X_0 = (x_0, y_0)$)
  - 새로운 직선 $\overrightarrow{XX_0} = X - X_0$
  - X, X_0에 대해 아래 만족
    $a(x - x_0) + b(y - y_0) = 0$
  - $N = (a, b)$, then
    $N \bullet (X - X_0) = 0$



출처: 네이버

참고

- 점과 직선 사이의 거리
  - 점 (x0, y0)과
    직선 ax + by + c = 0
    간의 거리
  - = $|ax_0+by_0+c| / \sqrt{a^2+b^2}$
  - 증명



$$\cos\theta = d / |\overrightarrow{PQ}|$$

$$\therefore d = |\overrightarrow{PQ}|\cos\theta = \frac{|\overrightarrow{PQ}|\,|\vec{n}|\cos\theta}{|\vec{n}|} = \frac{\overrightarrow{PQ}\cdot\vec{n}}{|\vec{n}|}$$

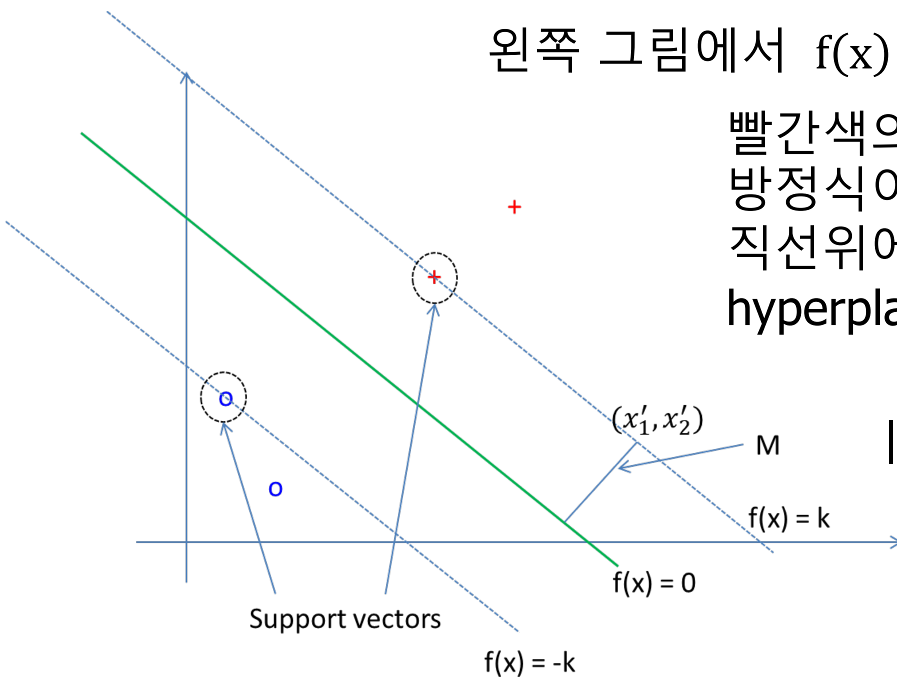
$$\overrightarrow{PQ} = (x_0 - x_1, y_0 - y_1),$$

- $\therefore \overrightarrow{PQ}\cdot\vec{n} = a(x_0 - x_1) + b(y_0 - y_1) = ax_0 + by_0 - (ax_1 + by_1)$
  $= ax_0 + by_0 + c$

$$|\vec{n}| = \sqrt{a^2 + b^2}$$

SVM

- Hyperplane이 $\mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = 0$으로 정의되는 경우 아래 그림과 같이 표현



왼쪽 그림에서 $\mathrm{f(x)} = \mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = 0$

빨간색의 support vector를 지나는 직선의 방정식이 $\mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = \mathrm{k}$, 해당 직선위에 있는 임의 점 (x_1', x_2')과 hyperplane 직선 간의 거리

$$|\mathrm{b}_0 + \mathrm{b}_1 x_1' + \mathrm{b}_2 x_2'| \Big/ \sqrt{b_1^2 + b_2^2}$$

Review

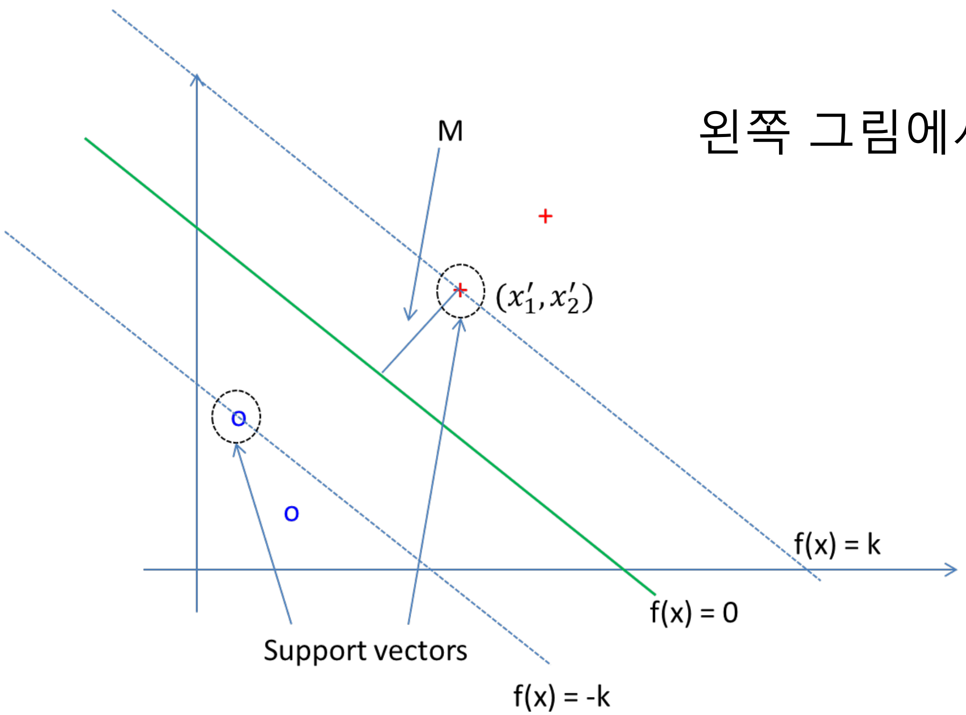
- SVM
  - 관측치를 공간상의 점으로 표현
  - 종속변수값 (즉, 클래스)에 따라 관측치들을 구분하는 hyperplane을 찾음
    - n 차원 공간에 대한 하이퍼플레인
      - $\mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 + \cdots + \mathrm{b}_\mathrm{n} x_\mathrm{n} = 0$
      - 예) 2차원 공간인 경우: $\mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = 0$
  - 여러개의 하이퍼플레인 존재, 이중 best 한 하이퍼플레인 선택
  - 서포트벡터하고의 거리(margin)를 최대화하는 하이퍼플레인

Review

- SVM (cont'd)
  - 서포트 벡터 중 하나가 (x_1', x_2') 이고, 하이퍼플레인을 $\mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = 0$ 으로 표현하는 경우,
    - Margin = $|\mathrm{b}_0 + \mathrm{b}_1 x_1' + \mathrm{b}_2 x_2'| \Big/ \sqrt{b_1^2 + b_2^2}$
    - 여기서$\mathrm{b}_0 + \mathrm{b}_1 x_1' + \mathrm{b}_2 x_2' = \mathrm{k}$ (또는$\mathrm{b}_0 + \mathrm{b}_1 x_1' + \mathrm{b}_2 x_2' = -\mathrm{k}$) 로 표현 가능 $\Rightarrow$ 이는 점 (x_1', x_2') 은 직선 $\mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = \mathrm{k}$ (또는 $\mathrm{b}_0 + \mathrm{b}_1 x_{s,1} + \mathrm{b}_2 x_{s,2} = -\mathrm{k}$)위의 점이라는 것을 의미
    - $\mathrm{f(x)} = \mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = 0$ 인 경우, 다음 페이지의 그림과 같이 표현 가능

Review

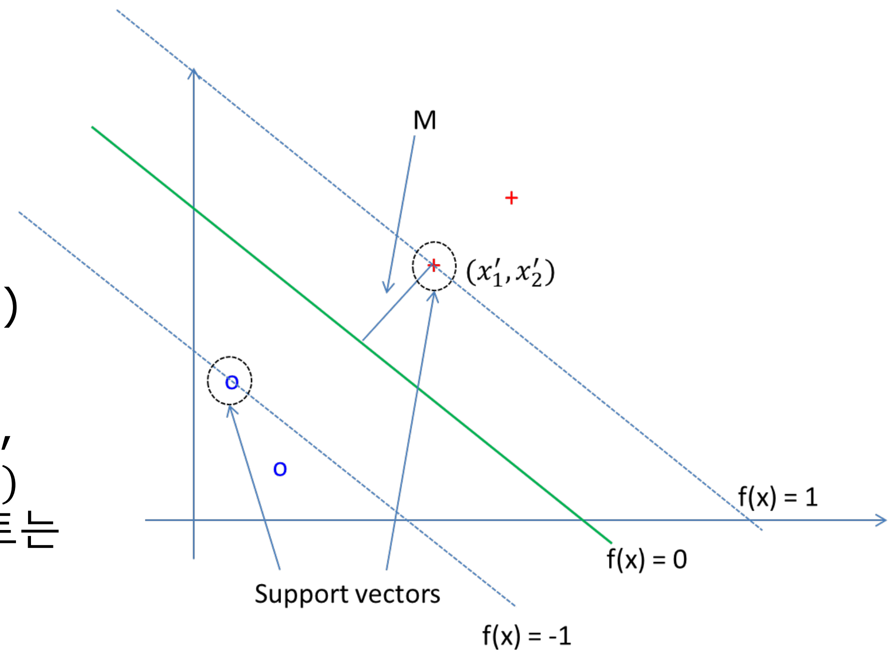
- SVM (cont'd)



왼쪽 그림에서 $\mathrm{f(x)} = \mathrm{b}_0 + \mathrm{b}_1 x_1 + \mathrm{b}_2 x_2 = 0$

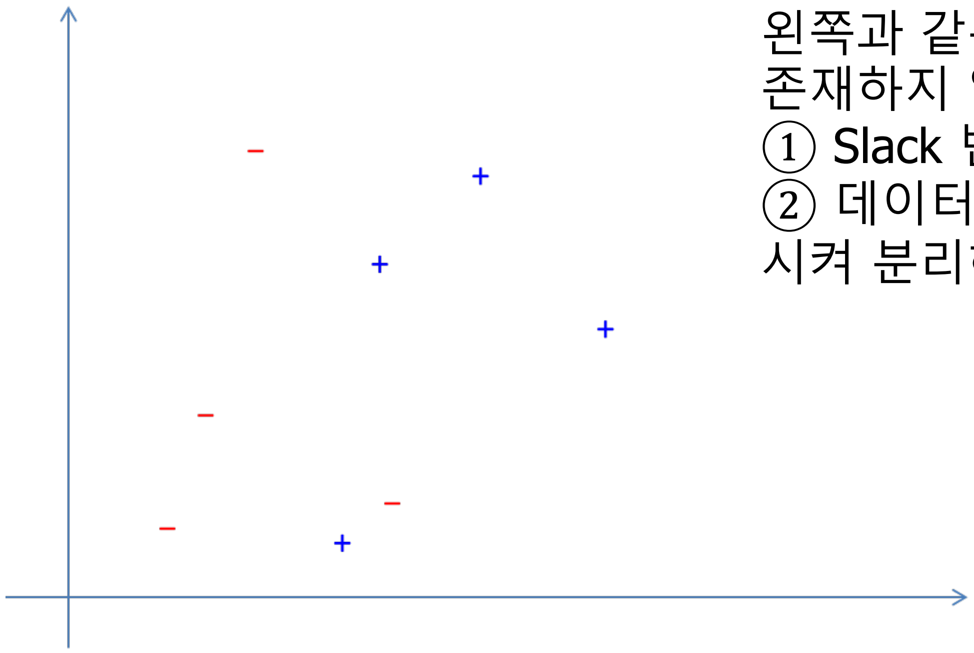
SVM

- 우리는 아래 최소화문제를 풀어야 함
  - $\underset{b_j}{\operatorname{argmin}} \sqrt{b_1^2 + b_2^2}$
- 추가 제약 조건
  - k=1 이라고 가정 (최소화문제와 상관없음)
  - $y_i \in \{-1, 1\}$ 라고 가정
  - 위의 그림에서 $\mathrm{f(x)} \geq 1$를 만족하는 점, 즉, 데이터 포인트는 긍정의 레이블을 $(y_i = 1)$ 갖고, $\mathrm{f(x)} \leq -1$을 만족하는 데이터 포인트는 부정의 레이블$(y_i = -1)$을 갖는 것, 즉,
  - $y_i f(x_i) \geq 1$
  - $\underset{b_j}{\operatorname{argmin}} \sqrt{b_1^2 + b_2^2}$, subject to $y_i f(x_i) \geq 1$
    - 참고: constrained optimization



SVM

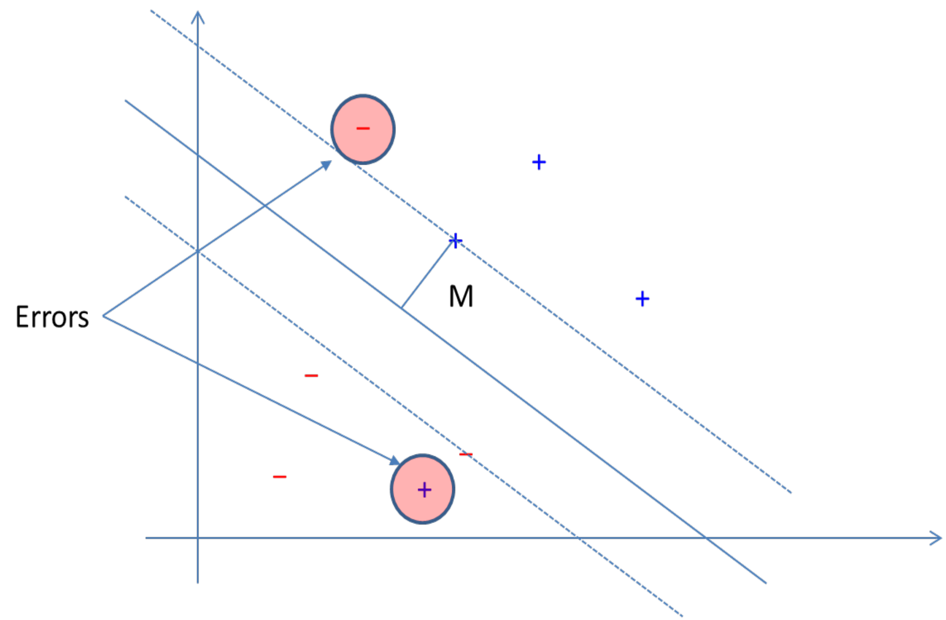
- Not linearly separable cases



왼쪽과 같은 경우, 직선 hyperplane은 존재하지 않는다. 이러한 경우,
① Slack 변수 사용하기
② 데이터 포인트들을 고차원 공간으로 이동시켜 분리하기

SVM

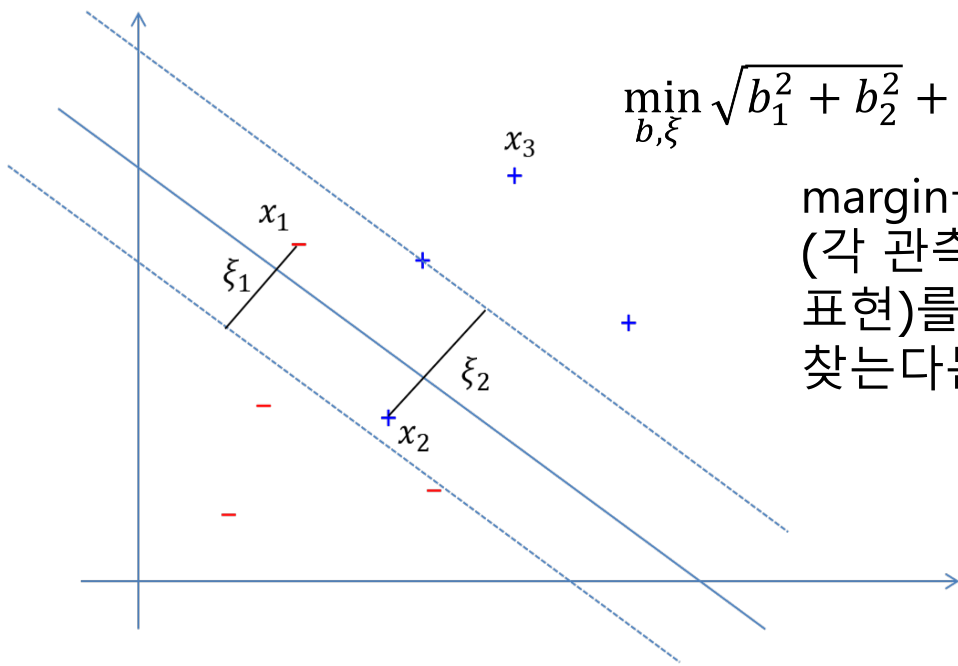
- 1. Slack 변수 사용하기
  - Slack 변수를 사용하는 방법은 어느 정도 에러를 인정하면서 분류를 하는 방법
    - 즉, hyperplane을 찾을 때 잘못 레이블링이 되는 관측치의 발생을 어느 정도 허용하는 방법
    - 이를 위해서 slack 변수 사용
    - Slack 변수는 각 관측치의 에러 정도를 나타내는 역할을 한다고 생각
      - y=1의 경우, f(x)=1 과의 거리
      - y=-1의 ㅣ 경우, f(x)=-1과의 거리



SVM

에러를 허용한다는 의미, 예) $y_i = 1$ 인경우, $f(x_i) = 1$로부터 ξ_i 떨어져 있을 수 있다라는 것을 의미

- 1. Slack 변수 사용하기 (cont'd)



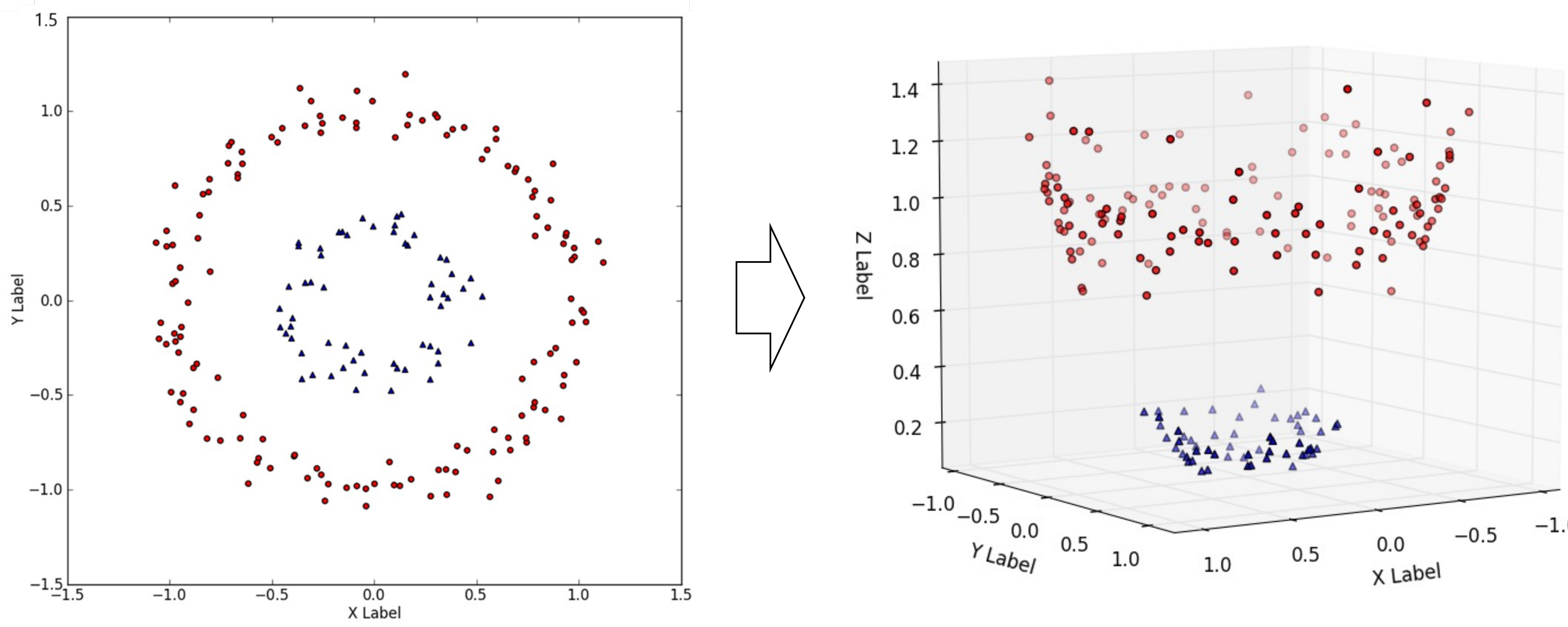
$$\min_{b,\xi} \sqrt{b_1^2 + b_2^2} + C \sum_{i=1}^{N} \xi_i, \text{ subject to } y_i f(x_i) \geq 1 - \xi_i$$

margin을 되도록 크게하면서 에러의 정도 (각 관측치들의 slack 변수값의 합으로 표현)를 최소화하는 hyperplane을 찾는다는 것을 의미

C의 의미: C의 값이 커지면, slack 변수의 값이 작아져야 한다는 것을 의미 ⇒ 즉, 에러를 조금만 허용 (하지만 과적합 문제 발생 가능성 증가)

SVM

- 2. 벡터를 고차원으로 이동하기 (kernel trick 사용하기)



SVM

- 2. 벡터를 고차원으로 이동하기 (cont'd)
  - 저차원의 데이터 포인트들을 고차원으로 이동시킨후 해당 공간에서 hyperplane을 찾는 과정에는 다음 두 단계가 필요
    - 1) 데이터 포인트들을 고차원으로 이동시키기
    - 2) 고차원에서의 데이터 포인트들 사이의 내적 계산하기
    - 하지만, 각 데이터 포인트를 고차원으로 직접 이동시킨 후, 내적을 연산하는 것은 시간이 너무 오래 걸림
    - 이를 해결하기 위해서 kernel function 사용 (이를 kernel trick라고도 함)
      - Kernel 함수를 사용하면, 점들을 직접적으로 고차원으로 보내지 않고 간단하게 고차원에서의 내적을 계산한 결과를 얻을 수 있음
      - 종류: 다항 kernel (polynomial kernel), rbf(radius basis function) kernel

SVM

- 다항 kernel function
  - $\mathrm{k}(\mathbf{x},\mathbf{y}) = (b + \gamma\mathbf{x}^T\mathbf{y})^d$
    - 이는 저차원의 두 벡터 (즉, $\mathbf{x},\mathbf{y}$)을 고차원으로 보낸 후의 내적 결과 리턴
    - 하지만, 직접적으로 고차원 공간으로 점들을 이동시키지는 않음
    - b와 γ (gamma), d는 하이퍼파라미터
    - 예) b = 0, γ = 1, d=2 이라고 가정
      - $\mathrm{k}(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$
      - $\mathbf{x} = (x_1, x_2), \mathbf{y} = (y_1, y_2)$ 인 경우
      - $\mathrm{k}(\mathbf{x},\mathbf{y}) = (x_1y_1 + x_2y_2)^2 = {x_1}^2{y_1}^2 + 2x_1x_2y_1y_2 + {x_2}^2{y_2}^2$
        - 이는 아래와 같은 두 벡터 간의 내적을 의미
        - $({x_1}^2, \sqrt{2}x_1x_2, {x_2}^2), ({y_1}^2, \sqrt{2}y_1y_2, {y_2}^2)$
        - k()는 2차원 공간에 있는 벡터들을 3차원 공간으로 이동 후 해당 공간에서 내적을 계산하는 역할
      - d의 값이 증가할 수록 더 높은 차원에서의 내적값
      - γ 값의 의미: 이 값이 커지면 고차원 공간에서의 두 벡터의 내적값이 커진다. 내적값은 유사도를 반영, 즉, 두 벡터가 더 유사한 벡터로 간주된다.

SVM

- rbf kernel function
  - $\mathrm{k}(\boldsymbol{x}, \boldsymbol{y}) = exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}\right)$
    - $exp(-\gamma\|\boldsymbol{x}-\boldsymbol{y}\|^2)$, $\frac{1}{2\sigma^2} = \gamma$
    - 여기에서 $\|\boldsymbol{x}-\boldsymbol{y}\|^2$는 두 벡터간 유클리디안 거리의 제곱을 의미
    - γ값의 의미: 이 값이 클수록 내적값이 작아짐, 즉, 유사도가 작아진다.

rbf kernel은 저차원 공간의 벡터를 무한 차원 (infinite dimension)으로 보내고 그곳에서의 내적을 구하는 효과가 있음. 자세한 내용은 https://www.youtube.com/watch?v=XUj5JbQihlU&t=1553s 을 참고

SVM

- Python code
  - See “SVC_example.ipynb”
  - 주요 하이퍼파라미터
    - C
    - kernel
    - gamma

Q & A

참고: CONSTRAINED OPTIMIZATION

Optimization with equality constraints

- Example:
  - $y = x_1 x_2 + 2x_1, where\ 4x_1 + 2x_2 = 60$
  - Find the values of x1 and x2 that maximize y.
- How to solve?
  - (Simply) 대입방법으로 풀 수 있다
    - $4x_1 + 2x_2 = 60$ => $x_2 = 30 - 2x_1$
    - 따라서 $y = x_1(30 - 2x_1) + 2x_1$
    - => x1 = 8, x2 = 14
    - 하지만 이 방법은 실제 분석에서는 사용되지 않는다.

Optimization with equality constraints

- Lagrange-multiplier method (라그랑주 승수법)
  - 이문제를 풀기위해 새로운 objective function을 생성 called the Lagrangian function
    - $Z = x_1 x_2 + 2x_1 + \lambda(60 - 4x_1 - 2x_2)$ (1)
      - where λ is called a Lagrange multiplier
      - If we can somehow be assured that $4x_1 + 2x_2 = 60$, so that the constraint will be satisfied, then the last term in (1) will vanish regardless of the value of λ
      - The questions is: How can we make the term vanish?

Optimization with equality constraints

- Lagrange-multiplier method
  - The tactic is to treat λ as an additional choice variable in (3), i.e., to consider $\mathrm{Z} = \mathrm{Z}(\lambda, x_1, x_2)$. Then the first order condition:
    - $z_\lambda = 60 - 4x_1 - 2x_2 = 0$
    - $z_1 = 0$
    - $z_2 = 0$
      - the first equation will automatically guarantee the satisfaction of the constraint
      - Thus, by incorporating the constraint into the Lagrangian function Z and by treating the Lagrangian multiplier as an extra variable, we can obtain the constrained extremum.

Optimization with equality constraints

- Lagrange-multiplier method
  - Generalization
    - $z = f(x, y)$, subject to the constraint $g(x, y) = c$
      - where c is a constant
    - We can write the Lagrangian function as
      - $Z = f(x, y) + \lambda[c - g(x, y)]$
    - The FOC becomes
      - $z_\lambda = c - g(x, y) = 0$
      - $z_x = f_x - \lambda g_x = 0$
      - $z_y = f_y - \lambda g_y = 0$
    - 그 다음 SOC을 추가적으로 test 해야함

Optimization with inequality constraints

- Similar to the optimization with equality constraints (Lagrange method with KKT conditions)
  - See https://machinelearningmastery.com/lagrange-multiplier-approach-with-inequality-constraints/
  - https://subprofessor.tistory.com/65