



CH5: Support Vector Machine

5.1 Support Vector Machine

5.1.1 Decision Boundary

5.1.2 SVM

5.1.3 Margin

5.1.5 Hard Margin

5.2 Soft Margin

5.2.1 Losses

5.2.2 Soft Margin SVM

5.3 Kernel Trick

5.3.1 Convex Sets

5.3.2 Lagrange Multiplier Method

5.3.3 Duality

5.3.4 KKT Condition

5.3.5 용어 정리

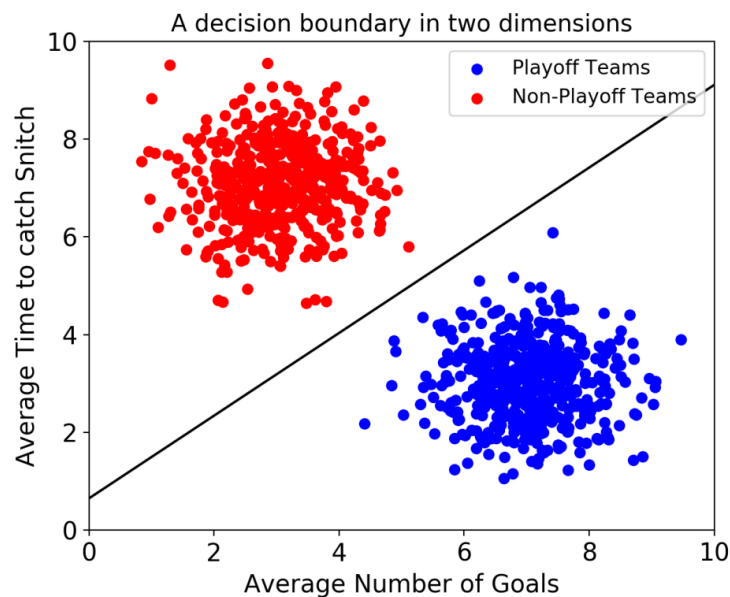
5.3.6 Dual problem & Representation of SVM

5.3.7 Mapping Functions & Kernel Function

5.3.8 Dual SVM with Kernel Trick

5.1 Support Vector Machine

서포트 벡터 머신(Support Vector Machine, SVM)은 기본적으로 이진 분류(Binary classification)을 위한 알고리즘이다. 머신러닝에서 사용되는 분류기는 결정 경계를 만들어 나가는 로직을 알고리즘마다 가지고 있다. 아래의 이미지는 SVM의 결정 경계 예시를 나타내고 있다.



SVM에 대해 본격적으로 알아보기 전에, 이 부분에서 사용되는 가정과 표기법에 대해 알아보자. 가장 먼저, 모든 실행(인스턴스)가 독립적이며 동일한 분포(iid)로 샘플링 되었다고 가정하자. SVM은 데이터를 라벨링하는데 있어서 일반적인 이진 분류기와 다른 점이 있다. 대부분의 이진 분류기는 1(Positive), 0(Negative)로 표기하는데, SVM은 -1(Negative), 1(Positive)로 클래스를 분류한다. 이렇게 표기하는 이유는 SVM의 알고리즘과 관련이 있는데, 이에 대해서 자세히 알아보자.

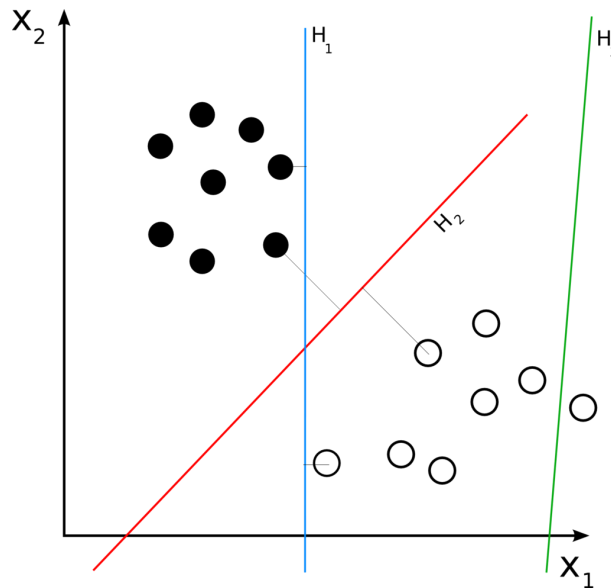
5.1.1 Decision Boundary

결정경계를 정하기 위한 알고리즘은 다음과 같다.

```
w · x + b = 0

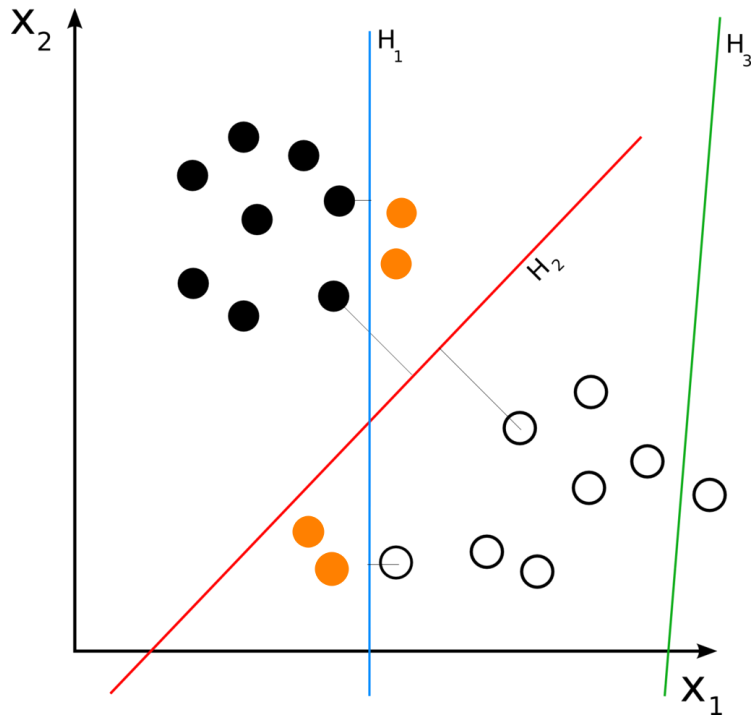
Positive case:
    w · x + b > 0
Negative case:
    w · x + b < 0
Confidence Level:
    (w · x_j + b) / ||w||
```

대부분의 경우에 선형 분류기의 결정 경계가 하나로만 나오지 않는다. 아래의 예시를 살펴보자.

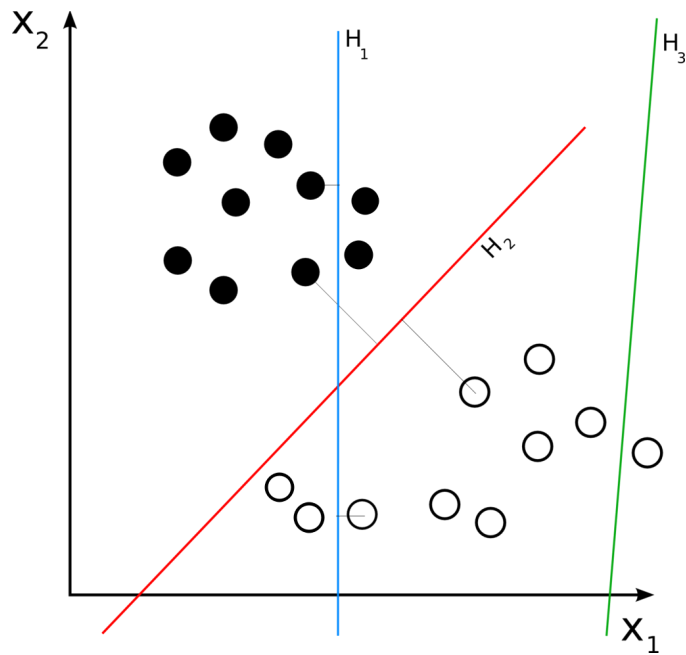


위에 H_1, H_2, H_3 라는 3개의 분류기가 있다. 일단 H_3 은 좋은 분류기라고 하기엔 무리가 있어 보인다. 총 16개의 인스턴스 중 검은색 클래스인 인스턴스가 8개, 흰색 클래스인 인스턴스가 8개가 있는데 H_3 은 8개의 검은색과 7개의 흰색을 하나의 클래스로 분류할 수 있다. 이를 엔트로피 등의 지표를 사용하면 H_3 이 좋지 않은 분류기임을 알 수 있다.

그렇다면 H_1 과 H_3 중에서는 어떤 분류기가 더 좋은 분류기일까? 두 분류기 모두 각자의 클래스에 맞게 분류를 잘 했기 때문에 주어진 인스턴스만으로는 두 분류기의 성능 비교를 할 수 없다. 그래도 굳이 비교해보자면, H_2 가 더 좋아보인다. '좋아 보이는 분류기의 비밀'은 바로 결정 경계(Decision boundary) 위에 있다는 것입니다.



아까전에 본 예시에서 4개의 인스턴스가 추가 되었다. 이 4개의 인스턴스는 H_1, H_2 중 어떤 분류기를 사용하여 분류하는지에 따라 클래스가 달라진다. 그림에서도 보이다시피, 아래쪽 2개는 흰색, 위쪽 2개는 검은색으로 분류될 거 같이 보인다. 인스턴스의 클래스가 실제로도 위쪽 2개는 검은색 클래스, 아래쪽 2개는 흰색 클래스라고 해보고 이를 그리면 다음과 같다.

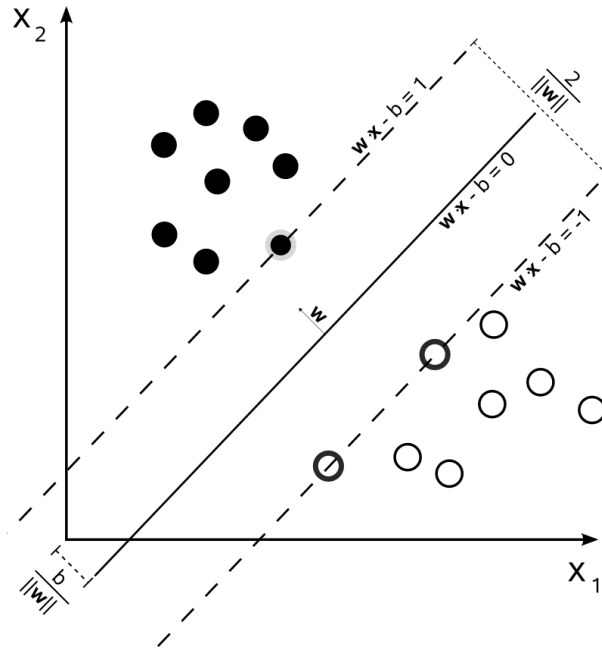


이제는 H_2 보다는 H_1 이 확실히 인스턴스를 잘 분류하고 있다는 것을 확인할 수 있다.

5.1.2 SVM

서포트 벡터 머신은 위 그림에서 빨간색으로 나타나는 분류기 H_2 를 찾기 위한 알고리즘이다. 서포트 벡터 머신이 결정 경계를 찾는 방식은 위에서도 설명했지만, 다시 살펴보자면 다음과 같다.

- 1) 각 클래스에서 앞쪽에 위치한 3개의 벡터를 찾는다.
- 2) 3개의 벡터는 각 클래스에서 가장 앞에 있는 벡터, 랜덤하게 선택하는 등 방법은 많다.
- 3) 3개의 벡터를 찾으면 같은 클래스에 속해 있는 2개의 벡터를 찾아 이를 연결하는 하나의 직선을 긋는다.
- 4) 이 직선과 평행하면서 나머지 하나의 벡터를 지나는 직선을 긋는다
- 5) 평행선을 중점을 지나며 평행선과 평행한 하나의 직선을 긋는다 (결정 경계)



5.1.3 Margin

결정된 결정 경계(실선)로부터 양쪽 직선까지의 거리를 마진(Margin)이라 한다. 결정 경계는 위에서 말했듯이, positive case 인 경우 $w \cdot x + b > 0$ 으로 분류하고, negative case인 경우 $w \cdot x + b < 0$ 으로 분류된다. 그리고 각각의 인스턴스의 실제 클래스는 y_i 라고 하겠다. 앞서 말했듯이, SVM는 신뢰도 $(w \cdot x + b)y_i$ 를 최대화하는 것이 목표이다.

$$\arg \max \sum_i (w \cdot x + b)y_i$$

신뢰도에서 y_i 값은 실제 클래스이므로 어떤 결정 경계를 택하더라도 변화가 없다. 그렇기 때문에 $w \cdot x + b$ 에 해당하는 마진을 최대화하는 것이 중요하다. 결정 경계 위에 있는 인스턴스를 x_p 라고 하면, 임의의 인스턴스 x 는 거리 r 를 사용하여 다음과 같이 나타낼 수 있다.

$$x = x_p + r \frac{w}{\|w\|}$$

$f(x) = w \cdot x + b$ 이면, $f(x_p) = 0$ 이므로, 위 식을 이용하여 r 에 대한 식으로 정리할 수 있다.

$$x = x_p + r \frac{w}{\|w\|}, f(x_p) = 0$$

$$f(x) = w \cdot x + b = w(x_p + r \frac{w}{\|w\|}) + b = wx_p + b + r \frac{w \cdot w}{\|w\|} = r\|w\|$$

$$\therefore r = \frac{f(x)}{\|w\|}$$

좋은 결정 경계를 구하기 위해서는, r 값을 최대로 늘려야 한다. 이때, 양 방향 모두 고려해야 한다. 서포트 벡터 x_s 에 대해서 $f(x_s) = w \cdot x_s + b = a$ 이므로 거리 r_s 를 아래와 같이 나타낼 수 있다.

$$r_s = \frac{a}{\|w\|}$$

즉, r_s 를 최대화 하는 w, b 는 $\|w\|$ 를 최소화하는 w, b 와 같게 된다. 이를 수식적으로 정리하면 다음과 같다.

$$\max_{w,b} 2r = \max_{w,b} \frac{2a}{\|w\|} = \min_{w,b} \|w\| \quad s.t. (wx_j + b)y_j \geq a, \forall j$$

이는 quadratic optimization이 된다.

▼ 🤔 궁금한 사람만 열어볼 것 🤔

Kernel Machine vs Neural Net

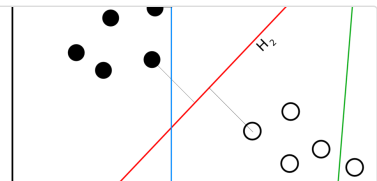
‘Margin이 클수록 더 좋은 분류기가 된다’와 ‘서포트 벡터 머신은 항상 마진이 최대가 되도록 한다’는 것을 기억하지...? 커널 머신이 잘 나가던 시절에 신경망, 즉 딥러닝이 공격받았던 지점이 이 부분이다. SVM은 항상 마진을 최대로 한다는 점, 즉 전역 최적점(Global Optimum)을 찾는다. 하지만 신경망은 경험적 위험만을 최적점으로 찾아가는 척도이기 때문에 지역 최적점(Local optimum)에 빠져버리기 쉽다.

하지만 2014년에 발표된 ‘Identifying and attacking the saddle point problem in high-dimensional non-convex optimization’에 의하면, 실험적으로 지역 최적점에 갇혀 빠져나가지 못하는 경우가 많지 않다는 것을 알아냈다. 저차원 공간에서는 지역 최적점에 빠졌을 때 경사 하강법 알고리즘으로 빠져 나오기가 어렵다. 하지만 실제 데이터는 수 십, 수 백 차원으로 구성되어있다. 이런 고차원 공간 상에서는 모든 방향으로 기울기가 0인 점이 거의 없다. 그렇기 때문에 학습을 충분히 한다면 경사 하강법으로도 전역 최적점을 찾아갈 수 있게 된다. 이 논문으로 인해 딥러닝은 오해를 풀고 연구에 박차를 가하게 되었으며 컴퓨터 비전이나 자연어처리 분야에서 각광받을 수 있었다.

서포트 벡터 머신(Linear & Hard Margin SVM)

해당 게시물은 고려대학교 강필성 교수님의 강의를 바탕으로 작성한 것입니다. 이번 게시물부터 이후 몇 개의 게시물에 걸쳐 커널 기반의 학습 방법의 대표격인 서포트 벡터 머신(Support Vector Machine, SVM)에 대해서 알아보겠습니다. 서포트 벡터 머신은 기본적으로 이진 분류(Binary classification)를 위한 알고리즘입니다. 머신

<https://yngie-c.github.io/machine%20learning/2021/03/07/svm/>

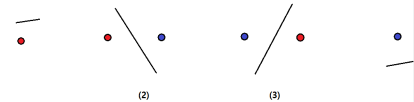


▼ 🤔 궁금한 사람만 열어볼 것 2: 커널 기반 학습 🤔

커널 기반 학습(kernel based Learning)

해당 게시물은 고려대학교 강필성 교수님의 강의를 바탕으로 작성한 것입니다. 커널 기반 학습(kernel based Learning)은 90년대부터 딥러닝이 대두되기 전인 2000년대 까지 머신러닝의 주류(?)에 있던 모델의 기반이 되는 학습 방법입니다. 본격적으로 커널을 사용한 모델을 알아보기 전에 커널 기반의 학습을 이해하는 데 필요한

<https://yngie-c.github.io/machine%20learning/2020/10/13/kernel/>



▼ 🤔 궁금한 사람만 열어볼 것 3: Linear programming & quadratic Programming 🤔

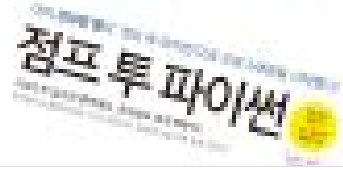
Quadratic Program(QP)는 목적함수(objective function)가 이차식(convex quadratic)이고, 제약함수(constraint functions)가 모두 affine인 convex optimization problem이다. General quadratic program은 다음과 같은 형태로 표현될 수 있다.

$$\min_x \frac{1}{2} x^T P x + q^T x + r \quad \text{subject to} \quad Gx \leq h, Ax = b$$

점프 투 파이썬

이 책은 파이썬이란 언어를 처음 접해보는 독자와 프로그래밍을 한 번도 해 본적이 없는 사람들을 대상으로 한다. 프로그래밍을 할 때 사용되는 전문적인 용어들을 알기 쉽게 풀어서 ...

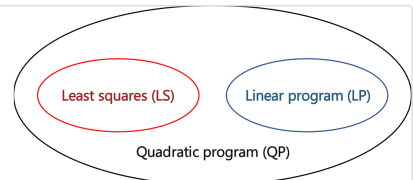
<https://wikidocs.net/17850>



[최적화] Quadratic program

저번 최적화 포스팅에 이어 이번에는 quadratic program을 정리해보려고 한다. 최적화의 모든 부분을 정리하고 싶지만 현실적으로 시간이 부족할 것 같아 앞으로도 지금처럼 주제별로 포스팅을 할 예정이고, 본 내용이 이 글을 읽는 분들에게 도움이 되었으면 좋겠다. 본문에서 다룰 내용은 다음의 세 가지 이다. 1. QP는 무엇이고, LP와

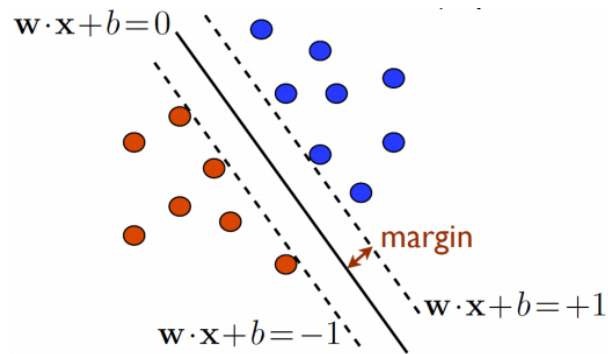
<https://velog.io/@wjleekr927/Quadratic-program>



5.1.5 Hard Margin

SVM은 결정 경계면이 선형인지 아닌지, 예외를 허용하는지에 따라 구분할 수 있다.

하드 마진(Hard margin)은 결정 경계면이 선형이며, 예외를 허용하지 않는다. 아래 식에서 x_i 는 각 인스턴스를 나타내는 y_i 는 인스턴스의 클래스를 나타낸다.

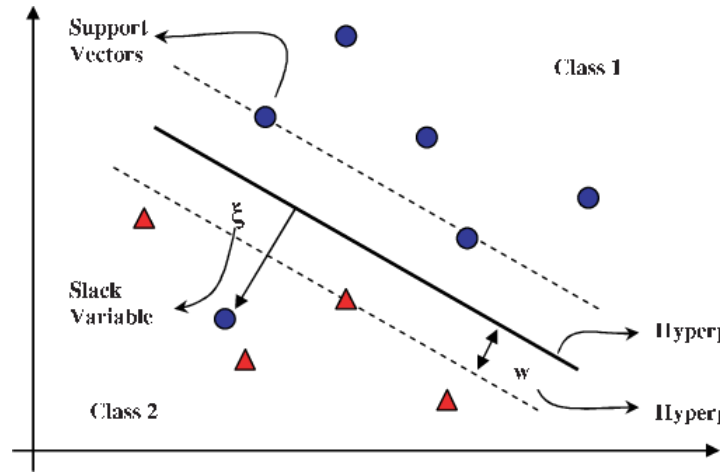


위 이미지에서 파란색으로 표현된 인스턴스의 클래스 $y_i = 1$ 이고, 모든 인스턴스는 $w^T x + b \geq 1$ 을 만족한다. 빨간색으로 표현된 인스턴스의 클래스 $y_i = -1$ 이고, 인스턴스의 $w^T x + b \leq -1$ 을 만족한다. 그렇기 때문에 모든 인스턴스가 $y_i(w^T x + b) \geq 1$ 라는 식을 만족한다. 이제 나머지 과정은 위와 같다.

5.2 Soft Margin

5.2.1 Losses

알다시피, 항상 모든 데이터를 두 쪽으로 분류할 수는 없다. 이런 경우를 에러(error)라고 부른다. 훈련 데이터가 아래와 같다고 해보자.



위 그림에서 class2(빨간색 세모)의 영역에 class1(파란색 동그라미)인 인스턴스가 위치해 있는데, 이를 선형 결정경계로 나누면, 이 인스턴스는 잘못 분류된 것으로 취급된다. 이런 경우, 인스턴스를 처리하는 방법은 크게 2가지가 있다.

- Option 1: 분류기를 벗어나는 Error case에 벌칙을 주는 penalization 방법
- Option2: 비선형 결정경계를 긋는 방법

이 두가지 방법은 error case에 벌칙을 주는 기준에 따라 두 가지로 나눌 수 있다. 먼저, error case에 해당하는 인스턴스의 개수를 기준으로 벌칙으로 가하는 방법이다. 이렇게 에러의 개수에 해당하는 error를 0-1손실(0-1 loss)이라고 하며, 식은 다음과 같다. 이때, C는 trade-off parameter이다.

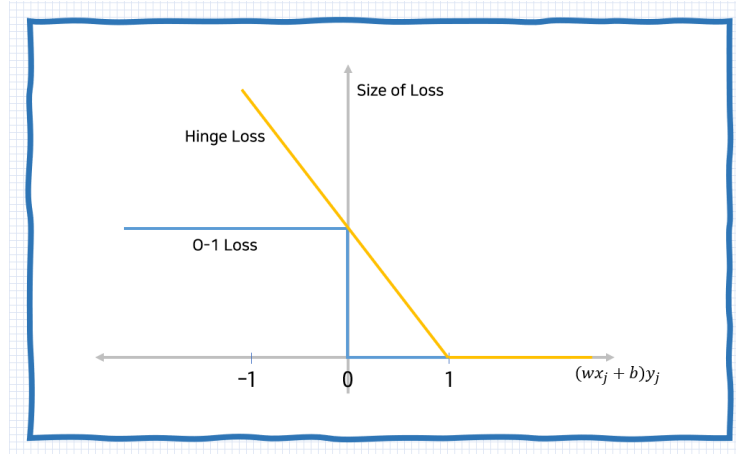
$$\min_{w,b} ||w|| + C \times \#_{error} \quad s.t(w x_j + b) y_j \geq 1, \forall j$$

하지만, 이 방법은 멀리 있는 점과 가까이 있는 점에게 모두 동일한 penalty값을 부여하기 때문에 error를 nice하게 정의할 수 없다. 따라서, 멀리 떨어진 error case 인스턴스에 더 강한 벌칙을 가하는 방법이 등장한다. 즉, slack variable, misclassification의 정도를 나타내는 변수를 사용하여 error를 구하는 것이다. 이렇게 멀리 떨어진 가중치를 구하는 손실을 힌지 손실 (Hinge Loss)라고 하고 ξ_j 로 나타낸다. 힌지 손실은 동일한 클래스의 서포트 벡터에서 0이며, 결정 경계에서는 1이 된다. 그리고, 결정 경계에서 반대 클래스 쪽으로 거리가 멀어질수록 선형적으로 loss가 증가하며, 식은 다음과 같다.

$$\min_{w,b} ||w|| + C \sum_j \xi_j$$

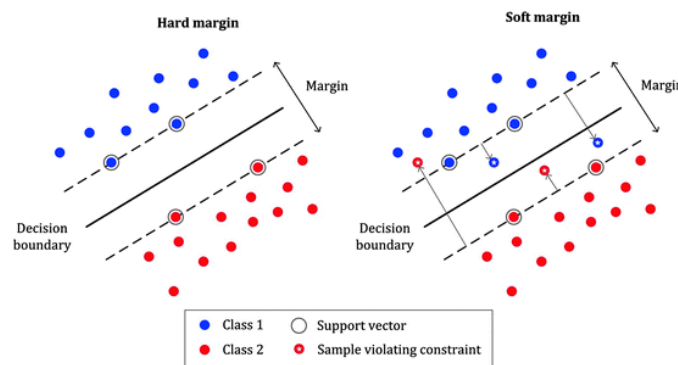
$$s.t (w x_j + b) y_j \geq 1 - \xi_j \forall_j, \quad \xi_j \geq 0 \forall_j$$

이를 그림으로 나타내면 다음과 같다. 0-1 손실은 파란색 선으로 나타나있고, 힌지 손실은 노란 선으로 나타나 있다. 실제로는 힌지 손실을 더 많이 사용하는데, 이는 0-1 손실은 quadratic으로 구현하기 어려우며 error의 정도를 반영하지 못하기 때문이다.

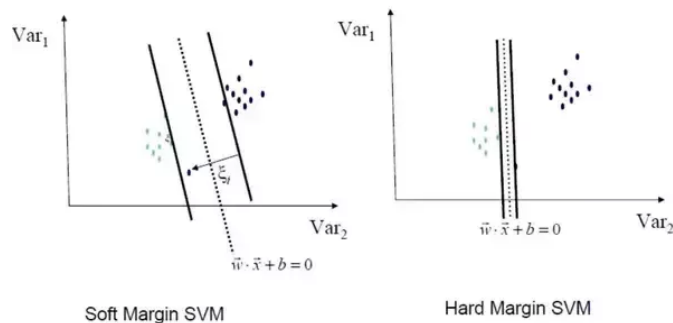


5.2.2 Soft Margin SVM

이렇게 에러는 고려하는 SVM을 소프트 마진 서포트 벡터 머신 (Soft margin SVM)이라고 한다. 반대로, 위에서 살펴봤던것처럼 하나의 오차도 허용하지 않는 모델을 하드 마진 서포트 벡터 머신 (Hard Margin SVM)이라고 한다.



하드 마진과 달리, 꼭 하나의 클래스에 같은 데이터가 들어있지 않아도 소프트 마진 서포트 벡터를 사용할 수 있다. 따라서, 소프트 마진 서포트 머신은 에러에 강건(Robust)하다는 장점이 있다. 아래의 왼쪽 그림은 데이터를 소프트 마진 SVM과 하드 마진 SVM으로 분류했을 때를 분류했을 때를 그림으로 나타낸 것이다.



오른쪽 하드 마진 같은 경우에는 모든 인스턴스가 같은 클래스에 속하게 하기 위해, 작은 결정 경계를 형성하게 된 것을 확인할 수 있다. 반대로, 소프트 마진의 경우, 에러 하나를 수용하여, 더 넓은 결정 경계를 가지는 것을 볼 수 있다.

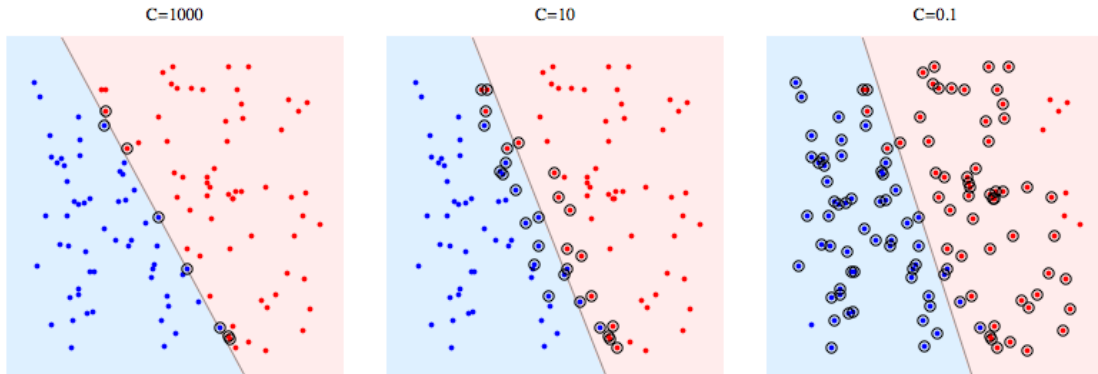
소프트 마진 SVM의 손실함수를 하이퍼파라미터 C를 조정하면, 에러를 얼마나 고려해 줄지를 결정할 수 있다. 앞서 상용한 힌지 손실 함수식으로 자세히 알아보자.

$$\min_{w,b} ||w|| + C \sum_j \xi_j$$

$$s.t (wx_j + b)y_j \geq 1 - \xi_j \forall_j, \xi_j \geq 1 \forall_j$$

C가 커질수록 힌지 손실 값 ξ_j 에 대하여 더 많은 벌칙을 가하게 되는데, 이때는 하나의 error case도 허용하지 않기 때문에 더 좁은 결정 경계가 형성된다. 반대로 C가 줄어들면, error case에 대한 벌칙이 줄어들기 때문에, Error에 강건한 결정 경계가 형성된다. 아래는 그에 대한 예시이다.

그러면 어떻게 C값을 결정해야 하는 걸까? 얼핏 생각해보면, C값이 크면 어느정도 합리적인 결정 경계가 형성되는 것 같지만, 항상 그런 건 아니니 grid seach나 bayesian optimization을 통해 수동으로 찾으면 된다.

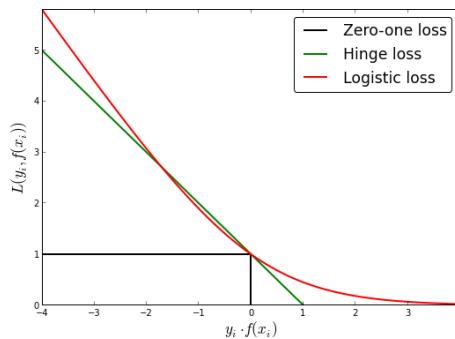


이제 SVM에 대해 알아봤으니 이를 logistic regression과 비교해보자. SVM은 앞서 말했듯이 hinge loss를 사용하고, 로지스틱 회귀는 log loss를 사용한다.

$$\hat{\theta} = \arg \max_{\theta} \sum_{1 \leq i \leq N} \log(P(Y_i|X_i; \theta)) = \arg \max_{\theta} \sum_{1 \leq i \leq N} \{Y_i X_i \theta - \log(1 + e^{X_i \theta})\}$$

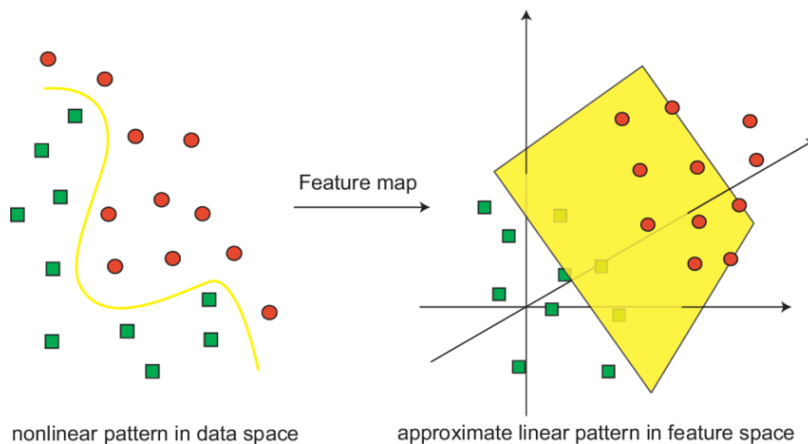
$$\xi_j = -\log(P(Y_j|X_j, w, b)) = \log(1 + e^{(wx_j + b)y_j})$$

이 둘의 관계를 그림으로 보면 다음과 같다. 결정 경계 근처에서 보면, log loss가 좋아보이나, overall을 보면 hinge loss가 더 좋다는 것을 확인할 수 있다.



5.3 Kernel Trick

에러를 허용하는 분류기 모델은 앞서 정의한 방법 중 하나일 뿐 항상 최선의 분류기라고 할 수 없다. 이럴 때는 비선형(Non-linear) 결정 경계를 그리는 SVM을 사용할 수도 있다.



기본적인 아이디어는 차원을 확장하여 두 데이터를 완전히 분류하는 방법과 동일하다. 하지만, SVM에서는 마구잡이로 데이터의 차원을 증가하면 계산량이 급격하게 증가며 많은 컴퓨팅 파워를 요구한다. *여기서 차원의 저주의 냄새가...* 이러한 문제를 해결하기 위해 등장한 것이 커널 트릭 (kernel trick)이다. 이 방법을 사용하면 계산량을 늘리지 않고도 non-linear한 결정 경계를 그릴 수 있다.

하지만 커널 트릭을 배우기 위한 여정은 길고 길기 때문에 하나씩 알아보자...



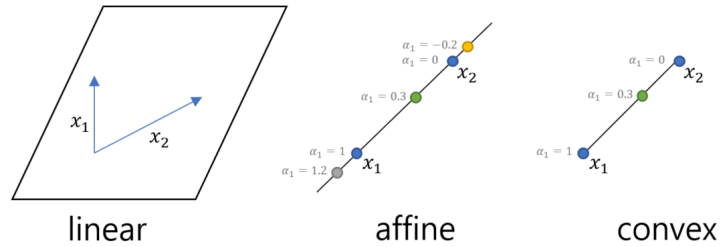
5.3.1 Convex Sets

가장 먼저 알아볼 것은 convex set과 convex function이다. Convex set은 다음과 같이 정의된다.

벡터 x_1, x_2, \dots, x_n 이 주어졌을 때, 이들을 결합하는 방법은 다음 3가지가 있다.

- Linear combination: $\alpha_1 x_1 + \dots + \alpha_n x_n$ (α_i 는 실수)
- Affine combination: $\alpha_1 x_1 + \dots + \alpha_n x_n$ ($\sum_i \alpha_i = 1$)
- Convex combination: $\alpha_1 x_1 + \dots + \alpha_n x_n$ ($\sum_i \alpha_i = 1$ 이고, $0 \leq \alpha_i \leq 1$)

Affine combination에 대해 닫힌 집합을 Affine set이라고 하고, 마찬가지로 convex combination에 대해 닫힌 집합을 convex set이라고 한다. 각각의 방법을 그림으로 나타내면 다음과 같다.

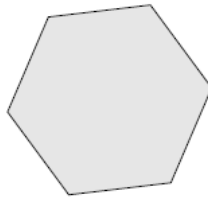


- linear combination: x_1 과 x_2 를 포함하는 R^2 의 평면이 span된다
- 2차원의 공간의 affine set은 x_1 과 x_2 를 지나는 직선이다.
- 2차원 공간의 convex set은 x_1 과 x_2 를 연결하는 선분이다. (벡터 앞에 붙는 계수가 부호가 중요)

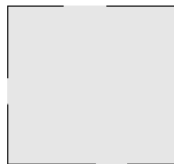
이제 convex set C에 대해 알아보자. Convex set은 다음과 같이 정의된다.

x 와 y 가 C에 속한다면, $tx + (1 - t)y$ 또한 C에 포함된다 ($0 \leq t \leq 1$)

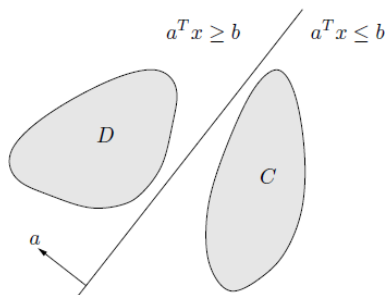
이를 예시를 통해 알아보자. 어떤 집합 C에 속한 임의의 두 점을 골랐을 때 둘을 연결하는 선분 또한 C에 포함될 경우, C를 convex set이라고 한다. 따라서 다음 도형은 convex set이다.



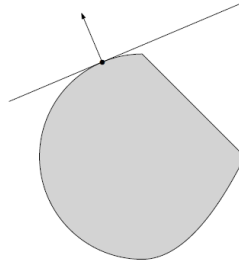
하지만, 두 점을 잇는 선분의 일부가 해당 집합 밖에 있거나, 임의의 두 점이 경계에 있을 경우 해당 점을 잇는 선분의 일부가 집합에 포함되지 않기 때문에 convex set이 되지 못한다.



Convex set에는 두 가지 중요한 성질이 있다. 첫번째는 separating hyperplane theorem이다. 두 개의 겹치지 않는 (disjoint) convex set을 분리해주는 하이퍼플레인 존재하며, 아래의 그림에서는 a 로 표기되어 있다. 여기서 D 와 C 는 닫힌집합이어야 하며, 둘 중 하나가 유계집합(bounded set)이어야 한다.

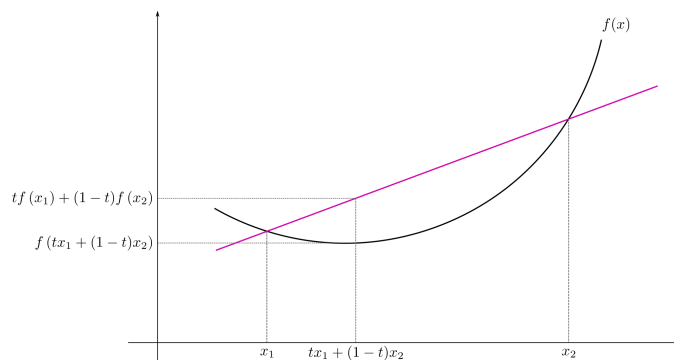


두 번째 성질은 supporting hyperplane theorem으로, convex set의 경계점을 지나는 접선이 항상 존재한다는 것이다.



Convex function은 임의의 두 점 x, y 와 $[0, 1]$ 사이의 값 t 에 대해 다음이 항상 성립하는 함수 f 를 가르킨다.

$$f(tx + (1 - t)y) \leq tf(x) + (1 - t)f(y)$$

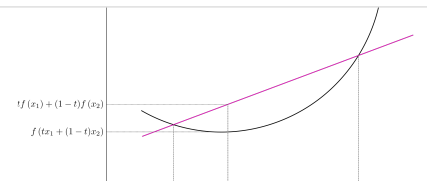


Convex function의 자세한 내용은 아래의 링크를 참고하자...!

Convex Functions

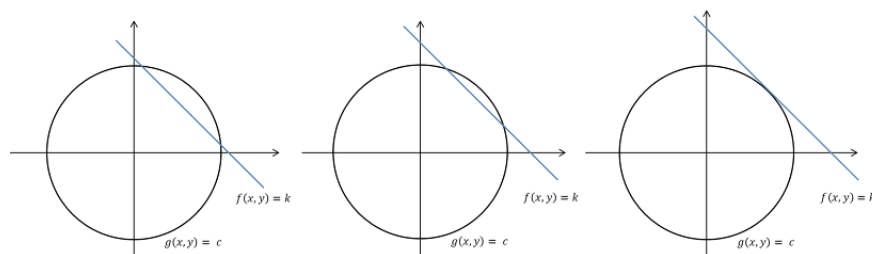
이번 글에서는 Convex Function(볼록함수)와 관련된 개념들을 살펴보도록 하겠습니다. 이 글은 미국 카네기멜런대학 강의 를 기본으로 하되 저희 연구실의 김해동 석사과정 이 만든 자료를 정리했음을 먼저 밝힙니다. 영문 위키피디아 또한 참고하였습니다. 그럼 시작하겠습니다. convex function 이란 임의의 두 점 x, y 와 $[0, 1]$ 사이의 값 t 에 대해 다음이 항상 성립하는 함수 f 를 가르킨다.

<https://ratsgo.github.io/convex%20optimization/2017/12/26/convexfunction/>



5.3.2 Lagrange Multiplier Method

먼저 라그랑주 승수법이란 최적화하려는 값에 형식적인 라그랑주 승수(multiplier)항을 더하여, 제약된 문제를 제약이 없는 문제로 바꾸는 방법이다. 라그랑주 승수법의 기본적인 아이디어는 어떤 조건 g 를 만족하는 함수 f 의 최대값 또는 최소값을 찾는 문제에서 찾고자 하는 값이 f 와 g 의 접점에서 존재할 수 있다는 가능성에서 나온 것이다.



위의 그림은 제약 조건 $g(x, y) = c$ 를 만족하는 $f(x, y)$ 의 최대값을 구하는 문제다. 만약 $f(x, y)$ 의 최대값을 k 라고 하면, k 는 기하학적으로 xy 평면에서 직선의 y 축 절편을 나타낸다. 따라서, 제약 조건 $g(x, y) = c$ 와 $f(x, y)$ 가 접할 때, $f(x, y)$ 는 최대가 된다.

즉, 라그랑주 승수법이란, 제약조건하에서 다변수함수의 최대, 최소를 구하는 방법인 것이다. 최소값의 경우 $\min f$ 로 표시하는데, 보통 최적화 문제를 풀 때는 임계점과 구간의 끝점에서 함수값을 비교해 해를 구하기 때문에 $\max f$ 로 쓴다. 라그랑주 승수법은 제약조건 하에서 최적화 문제를 해결하기 위한 방법으로 λ (Lagrange multiplier)를 사용한 함수 L 을 이용해 구할 수 있다. 아래의 식은 변수가 2개인 경우이다.

$$L(x, y, \lambda) = f(x, y) - \lambda(g(x, y) - c)$$

$$L_x = f_x(x, y) - \lambda g_x(x, y) = 0$$

$$L_y = f_y(x, y) - \lambda g_y(x, y) = 0$$

$$L_\lambda = c - g(x, y) = 0$$

이 보조함수를 최적화 하기 위해서는 x, y 에 대한 편미분을 진행해야 한다. 아래의 식이 0이 되는 경우의 (x, y) 가 두 함수가 접하는 최적의 (x, y) 가 되는 것이다.

$$\nabla L(x, y, \lambda) = \nabla f(x, y) - \lambda \nabla g(x, y) = 0$$

5.3.3 Duality

그 다음으로 알아볼 것은 duality, 즉 쌍대성이다.

최적화 이론에서 쌍대성이란, 어떤 최적화 문제가 원초문제(primal problem)와 쌍대문제(the dual problem)의 두 가지 관점에서 볼 수 있다는 원칙이다. 쌍대 문제의 상향은 primal problem의 하한이 된다. 결론적으로 말해, primal problem과 쌍을 이루는 것이 dual problem이다. Primal problem에서는 주어진 식을 만족하는 벡터 x 를 찾는 것이 목적이었으나, dual problem에서는 벡터 u, v 를 찾는 문제가 된다.

접근법은 많지만, 우리는 라그랑지안적 접근을 통해 더 자세히 알아보자 .

$$\min_x c^T x \quad \text{subject to } Ax = b, Gx \leq h$$

앞서 말했듯이, primal problem을 라그랑주 승수 벡터(Lagrange multiplier) u 와 v 를 통해 라그랑주 함수 L 을 만들면 아래와 같은 형태가 된다. 이때, $v \geq 0$ 이다.

$$L(x, u, v) = c^T x + u^T (Ax - b) + v^T (Gx - h) \leq c^T x$$

위의 식을 살펴보면, 앞선 조건 때문에 $Ax = b$ 는 항상 0이다. 마찬가지로, 앞서 설정한 제약조건 때문에 $Gx - h$ 는 0 이하의 값을 갖는다. 따라서 L 은 목적함수 $c^T x$ 보다 항상 작거나 같게 된다.

여기서 C 를 primal problem의 제약식을 만족하는 x 의 집합, f^* 을 우리가 찾으려는 최적의 값이라고 할때, 다음과 같이 식을 쓸 수 있다.

$$f^* \geq \min_{x \in C} L(x, u, v) \geq \min_x L(x, u, v) = g(u, v)$$

위의 식을 살펴보면, 제약조건이 있는 것보다는 없는 환경에서 식을 더 작게 만들수 있다. 이때, $g(u, v)$ 를 라그랑지안 듀얼 함수(Lagrange dual function)이라고 부른다. 그럼 $g(u, v)$ 은 어떤 값을 가질까?

우선 $g(u, v)$ 는 L 을 최소화하는 값인 $\min_x L(x, u, v)$ 이다. L 은 라그랑주 승수법에서도 알아봤듯이, 최소값을 찾기 위해서는 우리가 알고 싶은 미지수로 편미분한 값이 0이 되어야 한다. L 을 우리의 관심 미지수인 x 로 편미분한 결과는 다음과 같다.

$$\begin{aligned} \frac{\partial L}{\partial x} &= c^T + u^T A + v^T G = 0 \\ \therefore c &= -A^T u - G^T v \end{aligned}$$

이에 대한 dual problem($g(u, v)$)을 구하기 위해서는 위에 식을 L 에 대입하면 된다.

$$\begin{aligned}
L(x, u, v) &= c^T x + u^T (Ax - b) + v^T (Gx - h) \\
&= (-A^T u - G^T v)^T x + u^T (Ax - b) + v^T (Gx - h) \\
&= -u^T Ax - v^T Gx + u^T Ax - u^T b + v^T Gx - v^T h \\
&= -u^T b - v^T h \\
&= g(u, v)
\end{aligned}$$

앞서 이미 $f^* \geq g(u, v)$ 라는 관계를 확인했으므로, primal problem의 최소값을 찾는 것은 $g(u, v)$ 를 최대화하는 문제와 같다는 것을 알 수 있다. 단 라그랑주 승수법을 적용하는 과정에서 가정한 두 가지 조건인 $-A^T u - G^T v = c, v \geq 0$ 을 만족해야 한다. 따라서 해당 문제에 대한 dual problem은 다음과 같이 쓸 수 있다.

$$\max_{u, v} -b^T u - h^T v, \text{ subject to } -A^T u - G^T v = c, v \geq 0$$

라그랑지안 접근 방법은 지금까지 예시를 들었던 선형계획법 이외에도 임의적인 최적화 문제에 대해서도 모두 적용할 수 있다. 즉, 라그랑주 승수법을 통해 다양한 유형의 primal problem을 dual problem으로 바꿀 수 있다.

▼ ✨ 일반적인 케이스에 적용 ✨

Primal problem은 다음과 같다.

$$\min_x f(x) \text{ subject to } h_i(x) \leq 0, l_j(x) = 0$$

앞에서 봤듯이 부등식의 형태의 제약식에 붙는 라그랑주 승수 u_i 는 모두 0 이상의 값을 가져야 한다.

$$\begin{aligned}
L(x, u, v) &= f(x) + \sum_{i=1}^m u_i h_i(x) + \sum_{j=1}^r v_j l_j(x) \\
g(u, v) &= \min_x L(x, u, v)
\end{aligned}$$

이를 dual problem으로 바꾸면 다음과 같다.

$$\max_{u, v} g(u, v) \text{ subject to } u \geq 0$$

Duality

<https://ratsgo.github.io/convex%20optimization/2018/01/25/duality/>



앞서 $g(u, v)$ 는 f^* 의 하한이라고 했는데, 이는 dual problem의 목적함수를 최대화하는 것은 primal 목적함수를 최소화하는 문제를 뜻한다. 하지만, primal problem의 해와 dual problem의 해는 항상 같지 않다. 즉, f 의 최적값과 $g(u, v)$ 사이에 차이가 존재할 수 있다는 것이며, 이를 duality gap이라고 한다.

- Weak dual: $f^* \geq \min_x L(x, u, v) = g(u, v)$
- Strong dual: $f^* = \min_x L(x, u, v) \rightarrow$ duality gap이 없는 경우

5.3.4 KKT Condition

KKT(Karush-Kuhn-Tucker) 조건은 primal, dual solution과의 관계에서 도출된 조건으로, 최적화에서 이론과 실제 구현에서 핵심적인 역할을 한다. 여기서 사용되는 조건은 general problem으로 다음과 같은 primal problem을 가진다.

$$\min_x f(x) \text{ subject to } h_i(x) \leq 0, l_j(x) = 0$$

이를 바탕으로, KKT의 조건은 다음과 같다.

- Stationary(정상성, 시간에 영향 X): $0 \in \partial(f(x) + \sum_i u_i f_i(x) + \sum_{j=1}^r v_j l_j(x))$
- Complementary slackness: $u_i \cdot h_i(x) = 0$ for all i
- Primal feasibility: $h_i(x) \leq 0, l_j(x) = 0$ for all i, j
- Dual feasibility: $u_i \geq 0$ for all i

위의 4가지 조건을 바탕으로 다음과 같은 명제가 성립한다.

Necessity: If x^* and u^*, v^* are primal and dual solutions, with zero duality gap, then x^*, u^*, v^* satisfy the KKT conditions

Sufficiency: If

x^* and u^*, v^* satisfy KKT conditions, then x^*, u^*, v^* are primal and dual solutions

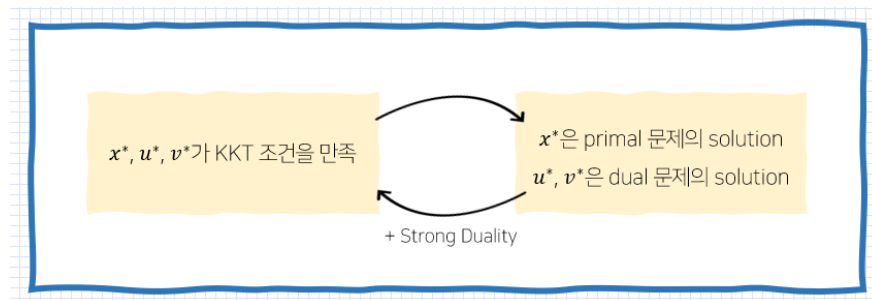
위의 조건에 대한 증명은 아래의 링크를 참고하자...! (더 쓰면 노션 멈출거 같아서 못쓰겠다...!)

KKT 조건

<https://ratsgo.github.io/convex%20optimization/2018/01/26/KKT/>



따라서, 해당 내용을 정리하자면 다음과 같다. 즉, Strong duality를 만족하는 경우 위 두 명제가 동치 관계를 갖는다.



5.3.5 용어 정리

이제 드디어 이 모든 것을 SVM에 적용할 때가 왔다.

먼저, SVM은 분류문제를 constrained quadratic programming으로 바꾼다. 이렇게 변형하면 좋은 이유는 앞서 말했듯이 차원이 늘어나는 문제를 방지하기 때문이다. 이제 진짜 시작하기 전에, 이 부분에서 사용될 notation을 정리하고 가자.

- **Lagrange method:** 라그랑주 승수법을 통해 다양한 유형의 primal problem을 dual problem으로 바꿀 수 있다
 - Lagrange Prime Function: $L(x, \alpha, \beta) = f(x) + \alpha g(x) + \beta h(x)$
 - Lagrange Multiplier: $\alpha \geq 0, \beta$
 - Lagrange Dual Function: $d(\alpha, \beta) = \inf_{x \in X} L(x, \alpha, \beta) = \min_x L(x, \alpha, \beta)$

- $\max_{\alpha \geq 0, \beta} L(x, \alpha, \beta) = \begin{cases} f(x) : \text{if } x \text{ is feasible} \\ \infty : \text{otherwise} \end{cases}$
- $\min_x f(x) \rightarrow \min_x \max_{\alpha \geq 0, \beta} L(x, \alpha, \beta)$

- **Primal Problem**

- $\min_x f(x) \text{ s.t. } g(x) \leq 0, h(x) = 0 \rightarrow \min_x \max_{\alpha \geq 0, \beta} L(x, \alpha, \beta)$

- **Lagrangian Dual Problem**

- $\max_{\alpha \geq 0, \beta} d(\alpha, \beta) \text{ s.t. } \alpha > 0 \rightarrow \max_{\alpha \geq 0, \beta} \min_x L(x, \alpha, \beta)$

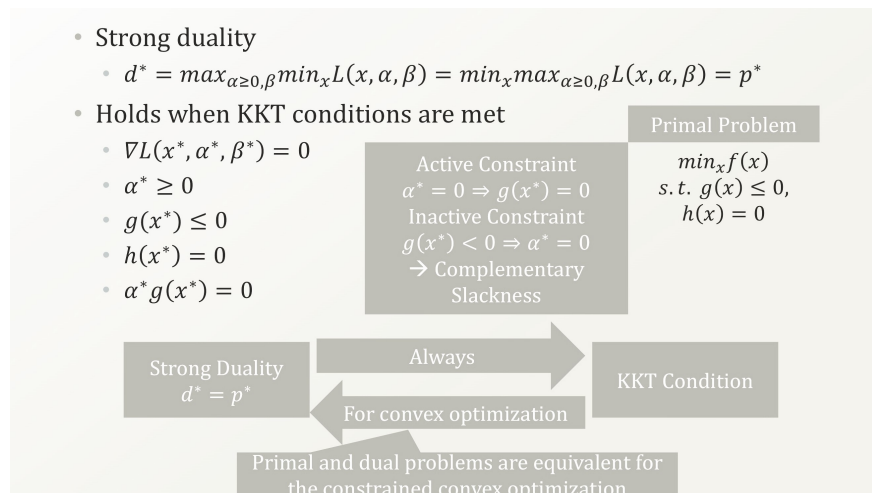
- **Weak duality theorem**

- $d(\alpha, \beta) \leq f(x^*) \text{ for } \forall \alpha, \forall \beta$
- $d^* = \max_{\alpha \geq 0, \beta} \min_x L(x, \alpha, \beta) \leq \min_x \max_{\alpha \geq 0, \beta} L(x, \alpha, \beta) = p^*$
 - Maximizing the dual function provides the lower bounds of $f(x^*)$
- Duality gap: $f(x^*) - d(\alpha^*, \beta^*)$

- **Strong Duality**

- $d^* = \max_{\alpha \geq 0, \beta} \min_x L(x, \alpha, \beta) = \min_x \max_{\alpha \geq 0, \beta} L(x, \alpha, \beta) = p^*$
- When KKT conditions are satisfied

정리하자면 다음과 같다.



5.3.6 Dual problem & Representation of SVM

선형의 경계선을 같은 SVM의 primal problem을 다음과 같이 정의할 수 있다.

$$\min_{w,b} \|w\| \quad s.t. (wx_j + b)y_j \geq 1, \forall j$$

$$\min_{w,b} \max_{\alpha \geq 0, \beta} \frac{1}{2} w \cdot w - \sum_j [(wx_j + b)y_j - 1] \quad s.t. \alpha_j \geq 0, for \forall j$$

위와 같은 상황이 있다면, 라그랑지안 프라임 함수인 $L(x, \alpha, \beta) = f(x) + \alpha g(x) + \beta h(x)$ 을 적용하여 아래와 같이 정리할 수 있다. 즉, 아래의 식은 우리의 상황에 맞게 라르랑주 함수를 정리한 것이다.

$$L(w, \alpha, \beta) = \frac{1}{2} w \cdot w - \sum_j \alpha_j [(wx_j + b)y_j - 1]$$

이제 여기서 라그랑지안 듀얼 함수를 적용하면, 아래와 같은 식을 구할 수 있다.

$$\max_{\alpha \geq 0} \min_{w,b} \frac{1}{2} w \cdot w - \sum_j \alpha_j [(wx_j + b)y_j - 1] \quad s.t. \alpha_j \geq 0, for \forall j$$

이제 듀얼 함수를 찾았는데, 우리는 Strong duality를 적용하기 위해 KKT 조건을 적용해야 한다. 앞서 정의한 조건과 같이 w 에 대한 미분과 b 에 대한 미분을 구하면 다음과 같은 식을 유도할 수 있다.

$$\frac{\partial L(w, \beta, \alpha)}{\partial w} = 0, \frac{\partial L(w, \beta, \alpha)}{\partial \beta} = 0, \alpha_i \geq 0, \forall i$$

$$\alpha_i ((wx_j + b)y_j - 1) = 0, \forall i$$

$$w = \sum_{i=1}^N \alpha_i y_i x_i, \sum_{i=1}^N \alpha_i y_i = 0$$

이제 앞서 정의한 라그랑지안 듀얼 함수를 적용한 공식을 전개해보자! 전개를 위해 KKT 조건을 적용을 할 것이다. 왜 잘 있는 식을 전개하는지 궁금할 수 있다. 이는 앞서 구한 SVM 모델에서는 ω 에 대한 식이었지만, 라그랑지안 프라임과 듀얼을 거쳐 α 에 대한 식으로 변했기 때문이다. 이제 이는 classification 문제가 아닌, 다시 α 에 대한 optimization이 되며, constrained quadratic programming 문제가 된 것이다...!

$$L(w, b, \alpha) = \frac{1}{2} w \cdot w - \sum_j \alpha_j [(wx_j + b)y_j - 1]$$

$$= \frac{1}{2} ww - \sum_j \alpha_j y_j w x_j - b \sum_j \alpha_j y_j + \sum_j \alpha_j$$

$$= \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i x_j - \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i x_i - b \times 0 + \sum_j \alpha_j$$

$$= \sum_j \alpha_j - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i x_j$$

만약, α_j 값이 알려진다면, 다음과 같은 식으로 optimization을 진행할 수 있다.

$$w = \sum_{i=1}^N \alpha_i y_i x_i$$

$$\alpha_i ((wx_j + b)y_j - 1) = 0$$

5.3.7 Mapping Functions & Kernel Function

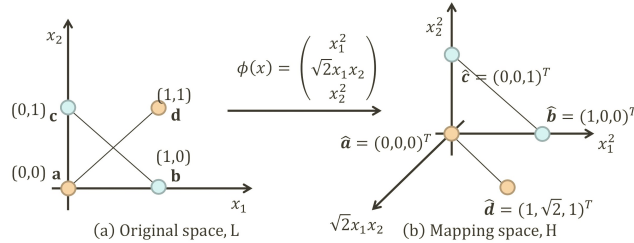
앞서 우리가 살펴본 상황은 선형경계를 가지는 SVM이었다. SVM은 두 범주를 잘 분류하면서 **마진(margin)**이 최대화된 **초평면(hyperplane)**을 찾는 기법이다. 즉, 기본적으로 선형분류를 한다. 하지만 어떤 직선을 그어도 두 범주를 완벽하게 분류하기 어려운 경우

도 많다. Kernel-SVM은 이 문제를 해결하기 위해 제안되었다. Kernel-SVM의 핵심 아이디어는 다음과 같다.

원공간(Input Space)의 데이터를 선형분류가 가능한 고차원 공간(Feature Space)로 매핑한 뒤 두 범주를 분류하는 초평면을 찾는 것이다.

따라서 우리는 커널 트릭을 적용해서 비선형 결정 경계를 만들 예정이다. 즉, x_1, x_2 의 interaction을 더욱 복잡하게 표현하는 것이다. Kernel을 도입하기 위해 SVM의 primal 문제를 dual문제로 바꿔야 하는데 이는 나중에 알아보자.

커널 트릭에서 차원을 저차원에서 고차원으로 변경하여 매핑하는 것을 매핑 함수(Mapping function, $\phi(x)$)이라고 한다. 아래는 매핑 함수로 2차원의 공간을 3차원으로 매핑한 예시이다.



위의 매핑 함수를 통해 저차원에서 고차원으로 이동한 커널함수에 대한 내적 값을 커널 함수로 나타내도록 정의했다.

The kernel calculates the inner product of two vectors in different space, preferably without explicitly representing the two vectors in the different space $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$

위의 예시에서 사용한 커널 함수는 polynomial kernel이었는데, 이 외에도 다양한 커널 함수가 존재한다.

- Polynomial(homogeneous): $k(x_i, x_j) = (x_i \cdot x_j)^d$
- Polynomial(inhomogeneous): $k(x_i, x_j) = (x_i \cdot c_j + 1)^d$
- Gaussian Kernel: $k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$
 - For $\gamma > 0$, sometimes parameterized using $\gamma = \frac{1}{2\sigma^2}$
- Hyperbolic tangent: $k(x_i, x_j) = \tanh(\kappa x_i \cdot x_j + c)$
 - For some $\kappa > 0$ and $c < 0$

▼ 🤔 그래서 왜 내적...? (Polynomial Kernel Function) 🤔

먼저 다음과 같은 x 와 z 값을 가지고 있다고 가정하자. $x = \langle x_1, x_2 \rangle$, $z = \langle z_1, z_2 \rangle$ 이 값들을 사용하여 다양한 커널을 적용해보자.

- Polynomial kernel function of degree 1

$$K(\langle x_1, x_2 \rangle, \langle z_1, z_2 \rangle) = \langle x_1, x_2 \rangle \cdot \langle z_1, z_2 \rangle = x_1 z_1 + x_2 z_2 = x \cdot z$$

- Polynomial kernel function of degree 2

$$K(\langle x_1, x_2 \rangle, \langle z_1, z_2 \rangle) = \langle x_1^2, \sqrt{2}x_1x_2, x_2^2 \rangle \cdot \langle z_1^2, \sqrt{2}z_1z_2, z_2^2 \rangle = x_1^2 z_1^2 + 2x_1 x_2 z_1 z_2 + x_2^2 z_2^2 = (x_1 z_1 + x_2 z_2)^2 = (x \cdot z)^2$$

- Polynomial kernel function of degree 3

$$K(\langle x_1, x_2 \rangle, \langle z_1, z_2 \rangle) = (x \cdot z)^3$$

- Polynomial kernel function of degree n

$$K(\langle x_1, x_2 \rangle, \langle z_1, z_2 \rangle) = (x \cdot z)^n$$

위의 예시에서 볼 수 있듯이, 2차원을 먼저 내적하나, n차원을 먼저 내적하나 결국 결과는 같다. 하지만 비용 측면에서는, 높은 차원의 내적을 먼저하는 것이 2차원의 내적을하고 n배 한것보다 비용이 많이 든다. 따라서, 커널을 사용하면, 고차원의 정보를 쉽게 다룰 수 있게 되는 것이다.

5.3.8 Dual SVM with Kernel Trick

이제 위의 커널을 사용해서 고차원과 비선형 결정경계를 가진 SVM의 dual problem을 구해보자. 아래의 식을 통해 classification을 할 수 있다. 식을 살펴보면, 위의 $\varphi(x_i)\varphi(x_j)$ 는 x 값들을 다른차원으로 보내고 있다. 이를 커널로 나타내면 $K(x_i, x_j)$ 가 되는데, 앞서 알아봤듯이 이는 내적을 사용하면 쉽게 계산할 수 있다...

$$\max_{\alpha \geq 0} \sum_j \alpha_j - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \varphi(x_i) \varphi(x_j) = \max_{\alpha \geq 0} \sum_j \alpha_j - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

아래의 w 를 보면, 방향성은 kernel을 통해 정할 수 없다는 것을 확인할 수 있다.

$$\begin{aligned} \alpha_i((wx_j + b)y_j - 1) &= 0, \quad C > \alpha_i > 0 \\ w &= \sum_{i=1}^N \alpha_i y_i \varphi(x_i) \\ b &= y_i - \sum_{i=1}^N \alpha_i y_i \varphi(x_i) \varphi(x_j) \end{aligned}$$

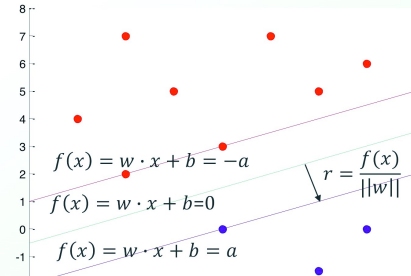
이 경우, 다음과 같이 방향을 정하면 된다.

$$\begin{aligned} \text{sign}(w \cdot \varphi(x) + b) &= \text{sign}\left(\sum_{i=1}^N \alpha_i y_i \varphi(x_i) \cdot \varphi(x) + y_j - \sum_{i=1}^N \alpha_i y_i \varphi(x_i) \varphi(x_j)\right) \\ &= \text{sign}\left(\sum_{i=1}^N \alpha_i y_i K(x_i, x) + y_j - \sum_{i=1}^N \alpha_i y_i K(x_i, x_j)\right) \end{aligned}$$

다른 linear case나 transformed case의 경우는 다음과 같다.

- Linear case
 - $\text{sign}(w \cdot x + b)$
 - $\min_{w,b} \|w\|$
 - $(wx_j + b)y_j \geq 1, \forall j$
- Transformed case
 - $\text{sign}(w \cdot \varphi(x) + b)$
 - $\min_{w,b,\xi_j} \|w\| + C \sum_j \xi_j$
 - $(w\varphi(x_j) + b)y_j \geq 1 - \xi_j, \forall j$
 - $\xi_j \geq 0, \forall j$
- Kernel trick case
 - $\text{sign}(w \cdot \varphi(x) + b)$
 - $\max_{\alpha \geq 0} \sum_j \alpha_j - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j)$
 - $w = \sum_{i=1}^N \alpha_i y_i \varphi(x_i)$
 - $b = y_j - w\varphi(x_j)$ when $0 < \alpha_j < C$
 - $\sum_{i=1}^N \alpha_i y_i = 0$
 - $0 \leq \alpha_i \leq C, \forall i$


Classification with SVM Kernel Trick

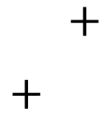


▼ 참고자료

초짜 대학원생의 입장에서 이해하는 Support Vector Machine (1)

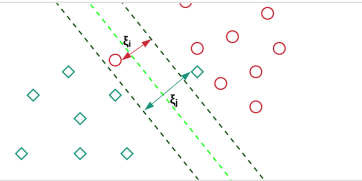
Lec. 16 Learning: Support Vector Machines, Patrick Winston MIT OCW 6.034 Fall 2010을 바탕으로 한 리뷰. 오늘은 클래식한 내용을 다뤄볼까 합니다. 기계 학습 분야에서 한 시대를 풍미하였고, 여전히 매우 다양한 현장에서 사용되고 있는 Support Vector Machine (SVM)에 대해 정리해보았습니다. SVM은 이미 역사가 오래

 <https://jaejunyoo.blogspot.com/2018/01/support-vector-machine-1.html>




소프트 마진(Soft Margin) SVM

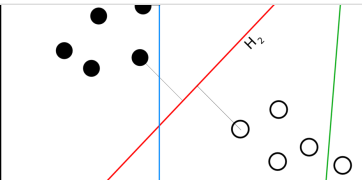
 https://yngie-c.github.io/machine%20learning/2021/03/13/soft_margin_svm/



서포트 벡터 머신(Linear & Hard Margin SVM)


해당 게시물은 고려대학교 강필성 교수님의 강의 를 바탕으로 작성한 것입니다. 이번 게시물부터 이후 몇 개의 게시물에 걸쳐 커널 기반의 학습 방법의 대표격인 서포트 벡터 머신(Support Vector Machine, SVM)에 대해서 알아보겠습니다. 서포트 벡터 머신은 기본적으로 이진 분류(Binary classification)을 위한 알고리즘입니다. 머신

 <https://yngie-c.github.io/machine%20learning/2021/03/07/svm/>



이건 꼭 읽어보기...!

KKT 조건

 <https://ratsgo.github.io/convex%20optimization/2018/01/26/KKT/>

