SUPPORT VECTOR MACHINE

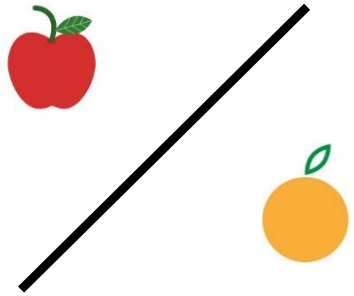
곽진주 박지은 임경룡



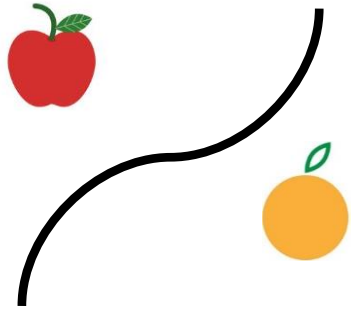
01

SVM?

Introduction



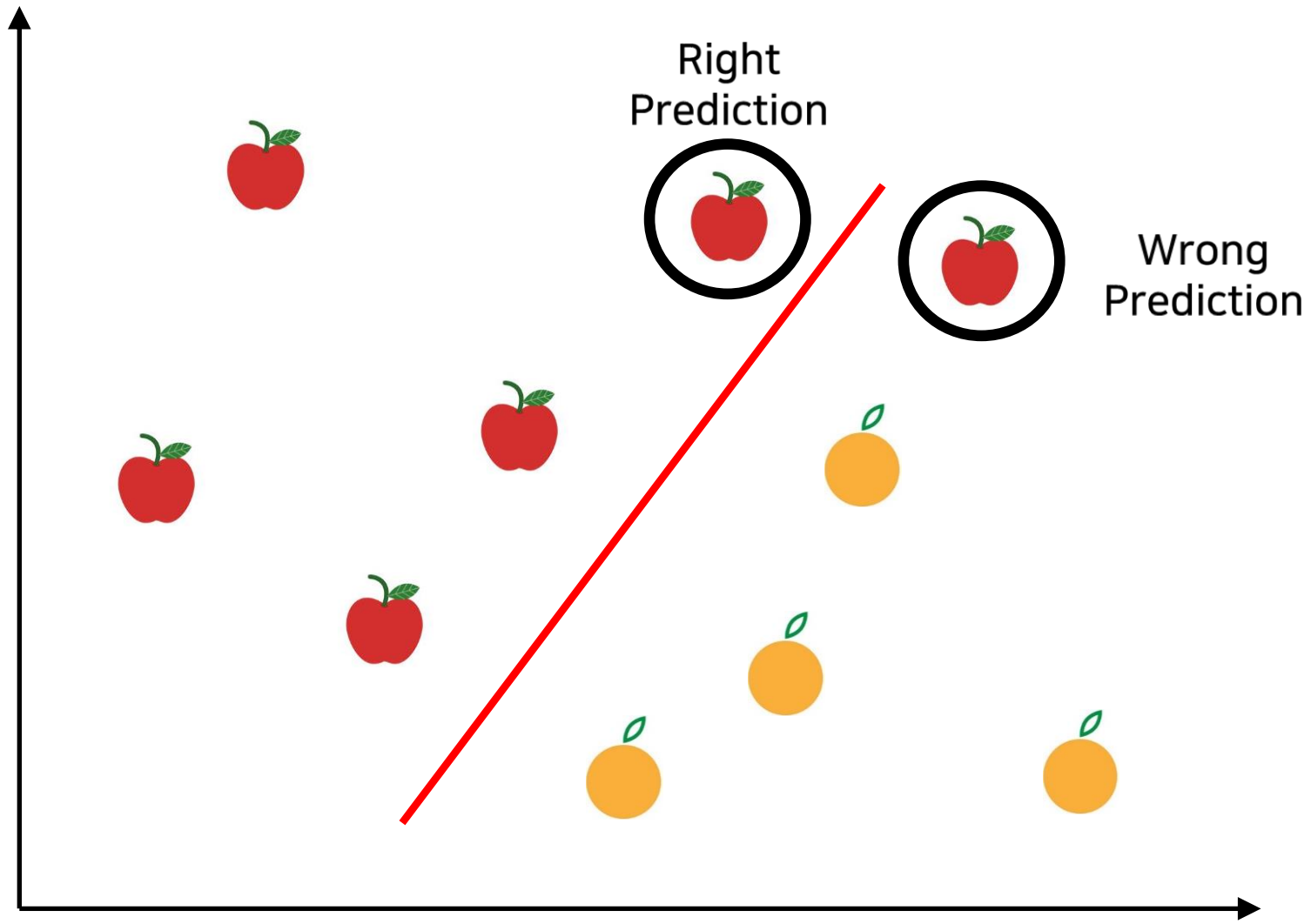
Linear SVM



Kernel SVM

과일을 분류해보자.

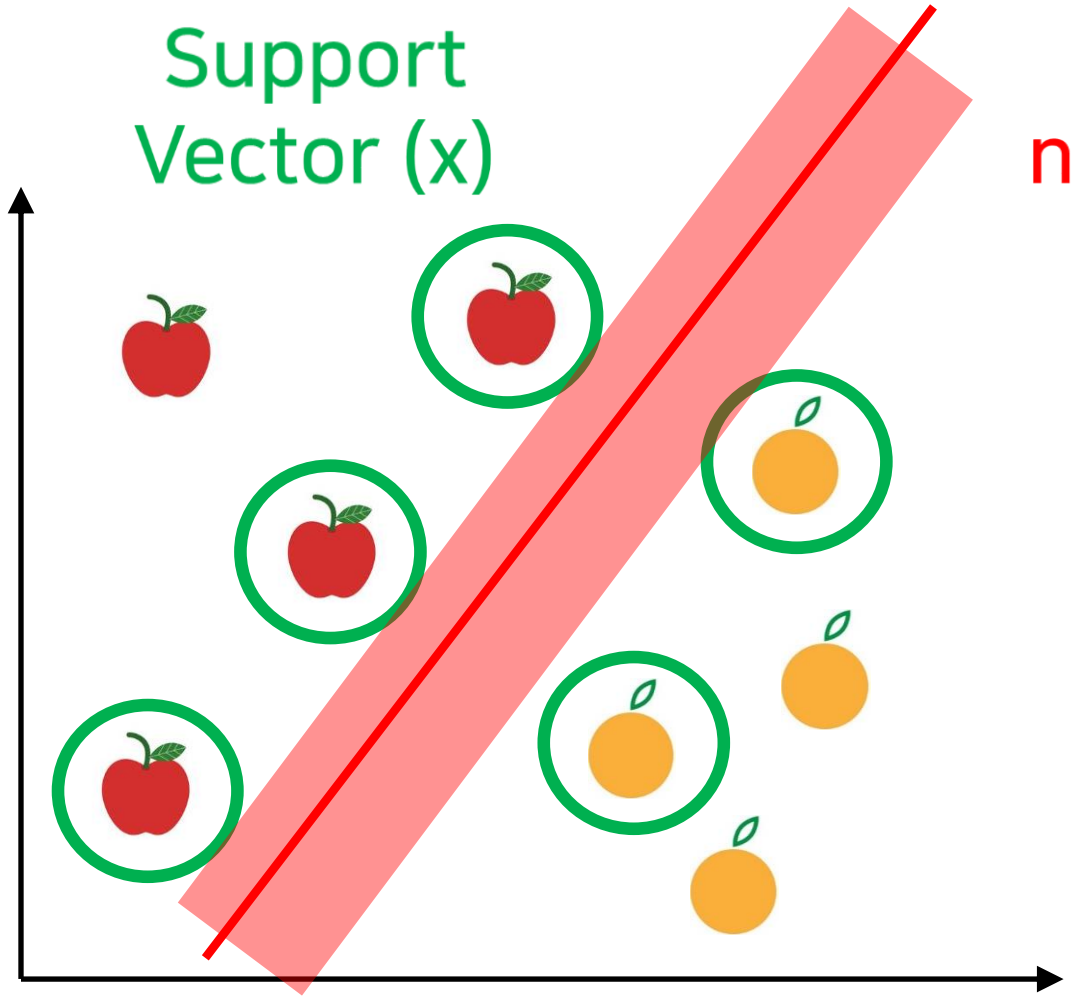
구분선이 최적임을
어떻게 확신하지?



Support Vector Machine?

초평면과 마진(Margin)을 통해 데이터를 분류하자!

Linear SVM 



n-1차원의
초평면

$$w^t x + b = 0$$

데이터(support vector)가
주어졌을때
w, b의 값을 찾자!

가장 좋은 초평면을 구하자!

가장 좋은 초평면?

Training data의 마진(Margin)을 최대화하는 것!

마진 (Margin) : 각 분류에서 가장 가까운 관측치 사이의 거리

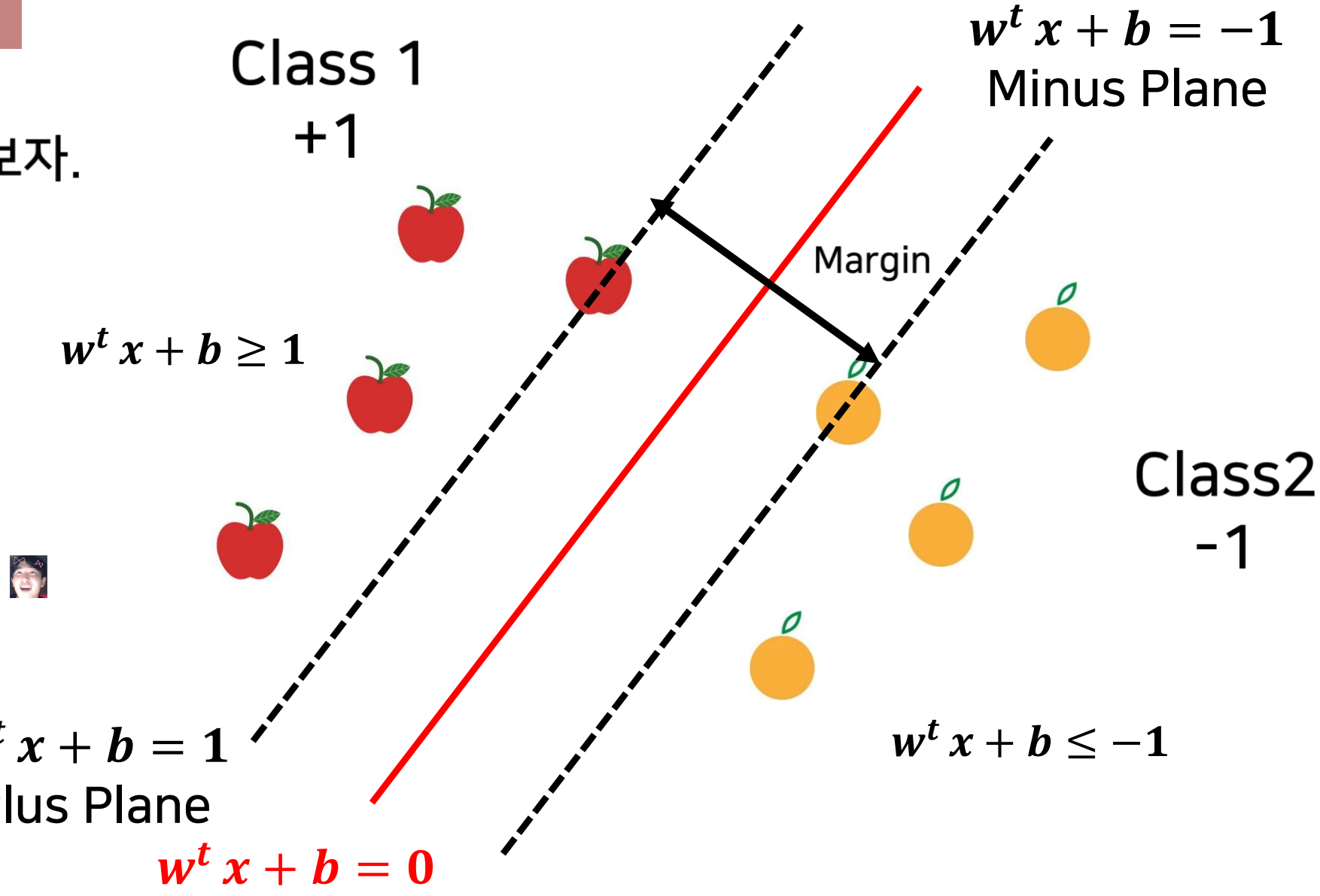
01

Margin

y = +1, -1 로 나누어보자.

$$x^+ = x^- + \lambda * w$$

평행이동의 관계가 있다!



Margin 

$$w^t x^+ + b = 1$$

Plus Plane 위의 점

$$w^t (x^- + \lambda w) + b = 1$$

$(x^+ = x^- + \lambda w)$ 대입!

$$w^t x^- + \lambda w^t w + b = 1$$

$-1 + \lambda w^t w = 1$ 이므로

$$\lambda = \frac{2}{w^t w}$$

$$\begin{aligned} Margin &= distance(x^+, x^-) \\ &= \|x^+ - x^-\|_2 \\ &= \|(x^- + \lambda w) - x^-\|_2 \\ &= \|\lambda w\|_2 \\ &= \lambda \sqrt{w^T w} \end{aligned}$$

3

$$= \frac{2}{w^T w} \cdot \sqrt{w^T w}$$

$$= \frac{2}{\sqrt{w^T w}} = \frac{2}{\|w\|_2}$$

$$\left(\sum_i |w_i|^2 \right)^{1/2} = \sqrt{w_1^2 + w_2^2 + \cdots w_n^2} = \sqrt{W^T W}$$

2차원 공간에서 벡터의 길이 : norm

Margin

최종목표 : Margin의 최대화 = $\frac{2}{\|w\|_2}$ 를 최대화하자!

$$max \ \frac{2}{\|w\|_2} = min \ \frac{\|w\|_2}{2}$$

$$\min \frac{1}{2}\|w\|_2 \Leftrightarrow \min \frac{1}{2}\|w\|_2^2$$

(제곱근 제거)

norm에 제곱근이 존재하므로
계산 편의를 위한 식 변경!

Margin 

$$\boldsymbol{min}\ \frac{\|\boldsymbol{w}\|_2}{\boldsymbol{2}}\quad subject\ to\ y_i\left(w^T x_i + b\right) \geq 1, i = 1,2, \ldots \ldots, n$$

y = +1, -1
각 Plain의 식을 기억하자!

어떻게 해결할까? : 라그랑주 승수법을 적용하자!

$$\max_{\alpha}\min_{w,b}\mathcal{L}(w,b,\alpha) = \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{n}\alpha_i(y_i(w^T x_i + b) - 1)$$

$$subject\ to\ \ \alpha_i \geq 0, i = 1,2, \ldots \ldots, n$$

라그랑주 승수법?

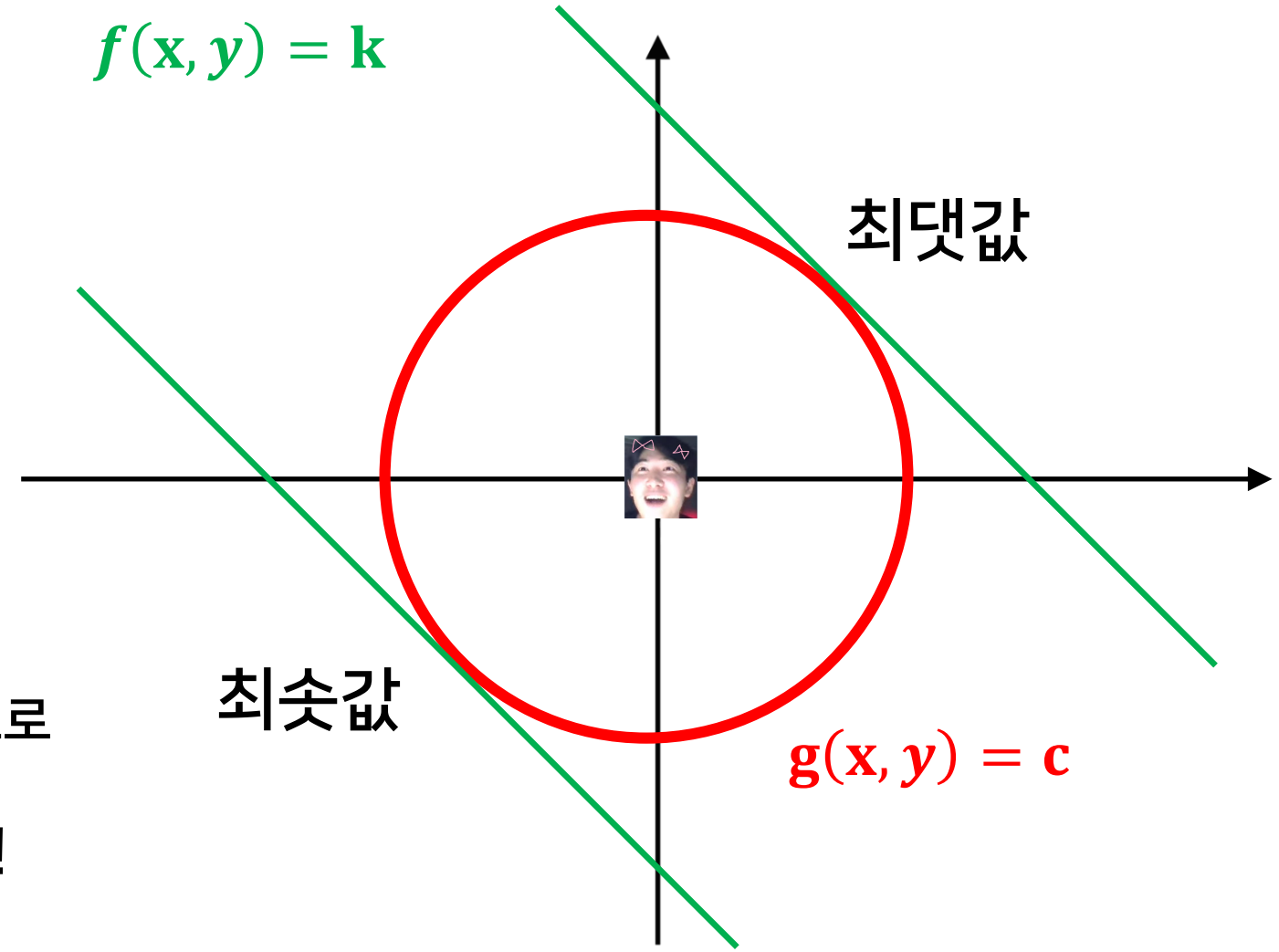
제약 조건 g를 만족하는 f의 최대, 최소를
f와 g의 접점에서 찾자!

라그랑주 승수법

기울기 벡터 $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$

접점에서 ∇f 와 접선 벡터의 내적은 0이므로

임의의 상수 λ 에서 $\nabla f = \lambda \nabla g$ 가 성립!



01

라그랑주 승수법

$\mathbf{g(x, y) = c}$ 일 때, 다음과 같은 함수가 정의된다.

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

$$\frac{\partial L(x, y, \lambda)}{\partial x} = 0 \qquad \frac{\partial L(x, y, \lambda)}{\partial y} = 0$$

L에 대한 각각의 기울기 벡터로
f와 g의 접점을 찾을 수 있다!

제약 조건이 있는 문제를
제약이 없는 식으로 변형하자

라그랑주 승수법

$$\max_{\alpha} \min_{w,b} \mathcal{L}(w,b,\alpha) = \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1)$$

$$subject\ to\ \ \alpha_i \geq 0, i = 1,2, \dots \dots, n$$

w,b에 대한 식 L에서 주어진 조건 (max, min)의
최적점을 찾기 위한 식

① $$\frac{\partial \mathcal{L}(w,b,\alpha)}{\partial w} = 0 \longrightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

② $$\frac{\partial \mathcal{L}(w,b,\alpha)}{\partial b} = 0 \longrightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$$

라그랑주 승수법

$$\underbrace{\frac{1}{2}\|w\|_2^2}_{①} \quad \underbrace{- \sum_{i=1}^{n} \alpha_i (y_i (w^T x_i + b) - 1)}_{②}$$

① $\frac{1}{2}\|w\|_2^2 = \frac{1}{2} w^T w$

$= \frac{1}{2} w^T \sum_{j=1}^{n} \alpha_j y_j x_j$

$= \frac{1}{2} \sum_{j=1}^{n} \alpha_j y_j (w^T x_j)$

$= \frac{1}{2} \sum_{j=1}^{n} \alpha_j y_j \left(\sum_{i=1}^{n} \alpha_i y_i x_i^T x_j \right)$

$= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$

RECALL

$$\frac{\partial \mathcal{L}(w, b, \alpha)}{\partial w} = 0 \longrightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

라그랑주 승수법

$$\underbrace{\frac{1}{2}\|w\|_2^2}_{①} \quad \underbrace{- \sum_{i=1}^{n} \alpha_i (y_i (w^T x_i + b) - 1)}_{②}$$

② $-\sum_{i=1}^{n} \alpha_i (y_i (w^T x_i + b) - 1)$

$$= -\sum_{i=1}^{n} \alpha_i y_i (w^T x_i + b) + \sum_{i=1}^{n} \alpha_i$$

$$= -\sum_{i=1}^{n} \alpha_i y_i w^T x_i - b \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i$$

$$= -\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n} \alpha_i$$

RECALL

$$\frac{\partial \mathcal{L}(w, b, \alpha)}{\partial b} = 0 \rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\frac{\partial \mathcal{L}(w, b, \alpha)}{\partial w} = 0 \rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

라그랑주 승수법

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{n} \alpha_i (y_i (w^T x_i + b) - 1)$$

$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

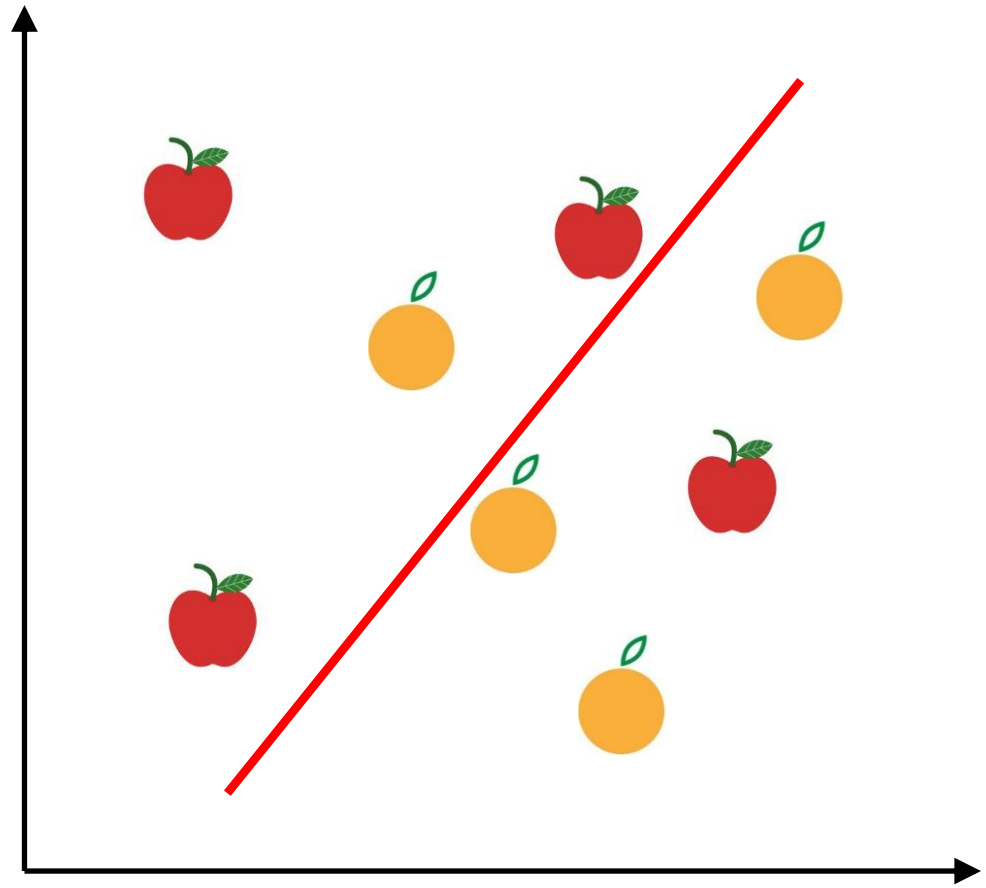
$$where \ \sum_{i=1}^{n} \alpha_i y_i = 0$$

Maximize!

α 값을 구하자

01

Nonseparable Problem



완벽한 선형 분리가 불가능

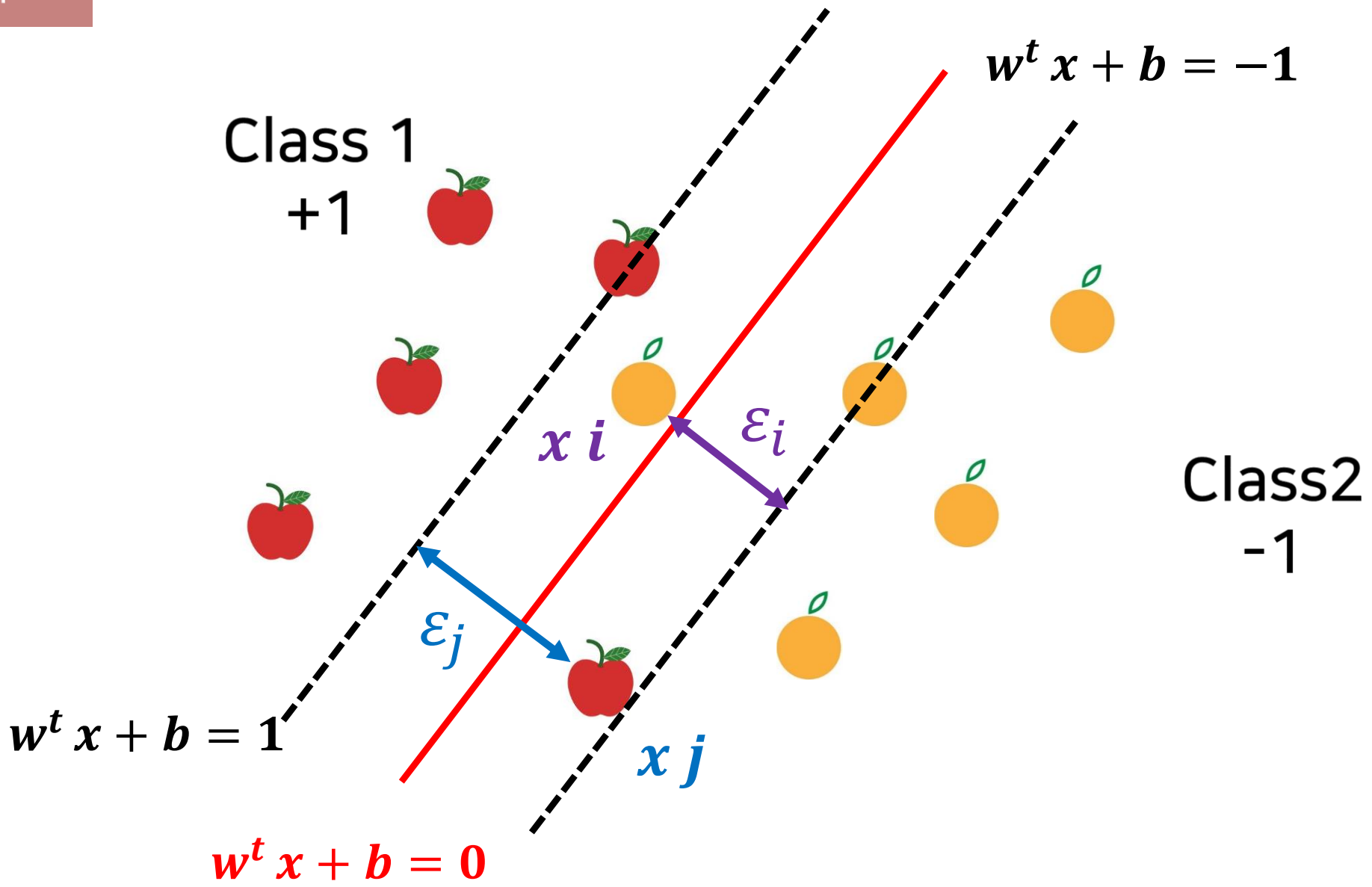
마진에 대한 융통성 ! 

SOFT Margin

01

Soft Margin

ξ 를 도입하여
오류 허용



Soft Margin

$$\underset{w,b,\xi}{minimize}\ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\xi_i$$

$$subject\ to\ y_i\left(w^T x_i + b\right) \geq 1 - \xi_i, \xi_i \geq 0, i = 1,2, \ldots\ldots, n$$

오류에 대한 조절이 필요 : 적절한 C (Cost) 를 통해 조절 

C 증가 : training error 많이 허용 x (overfit)
C 감소 : training error 많이 허용 (underfit)

Hard VS Soft

Hard Margin

$$\min_{w,b} \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{n} \alpha_i (y_i (w^T x_i + b) - 1)$$

$$= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

$$where \ \sum_{i=1}^{n} \alpha_i y_i = 0$$

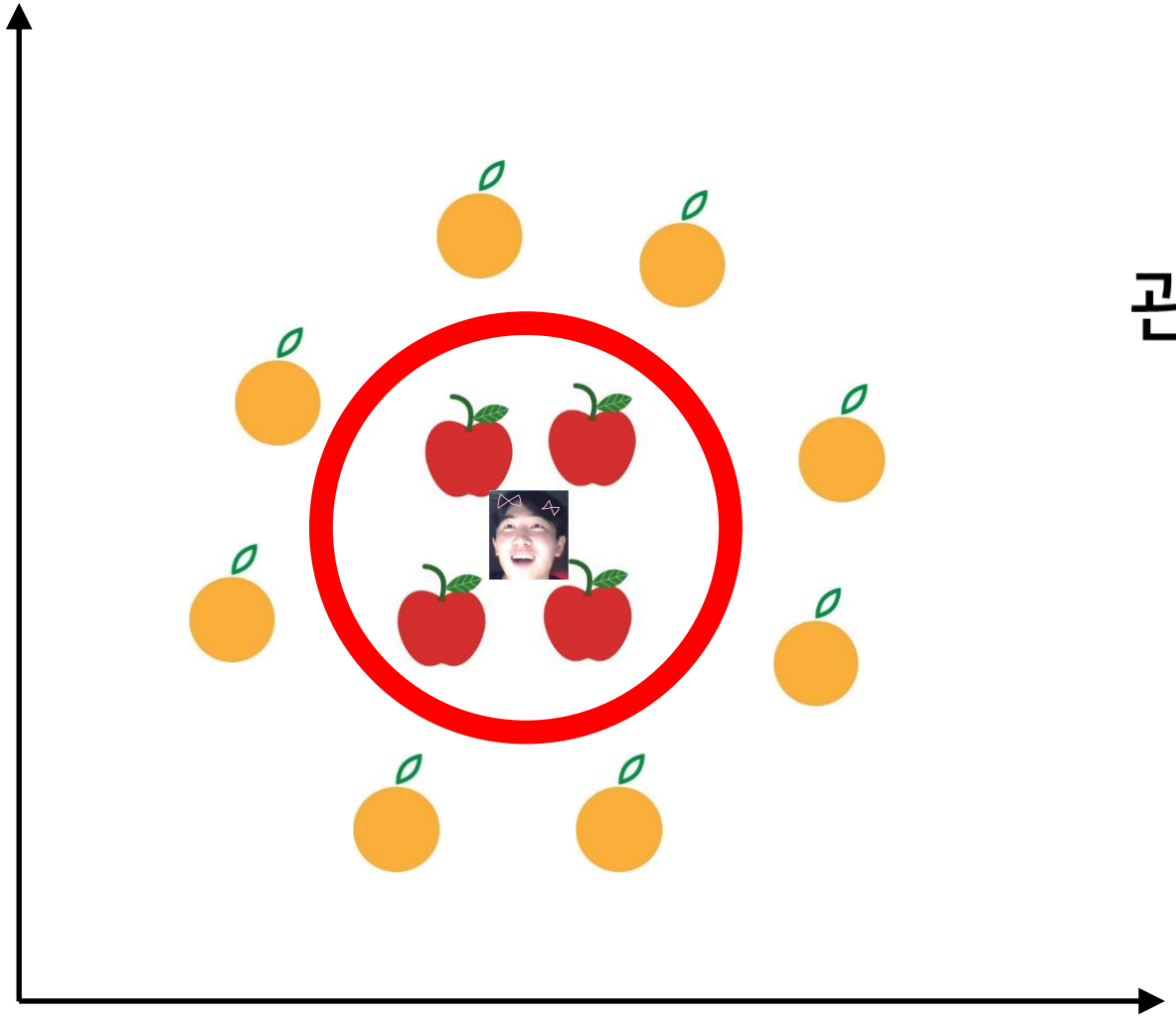
Soft Margin

오류 허용을 위한
Cost 사용!

$$0 \leq \alpha_i \leq C$$

01

Non-linear Problem



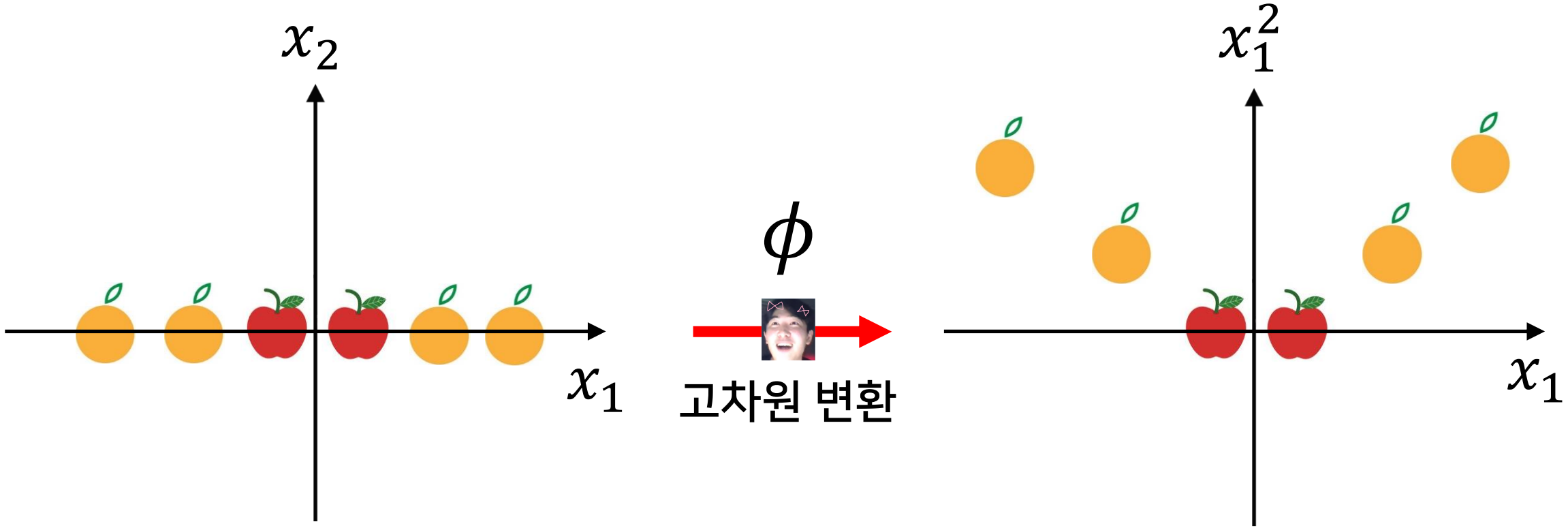
관측치 X를 더 높은 차원으로
변환하여 분류하자!

Kernel SVM

Kernel SVM

비선형 관계를 고차원으로 변환!

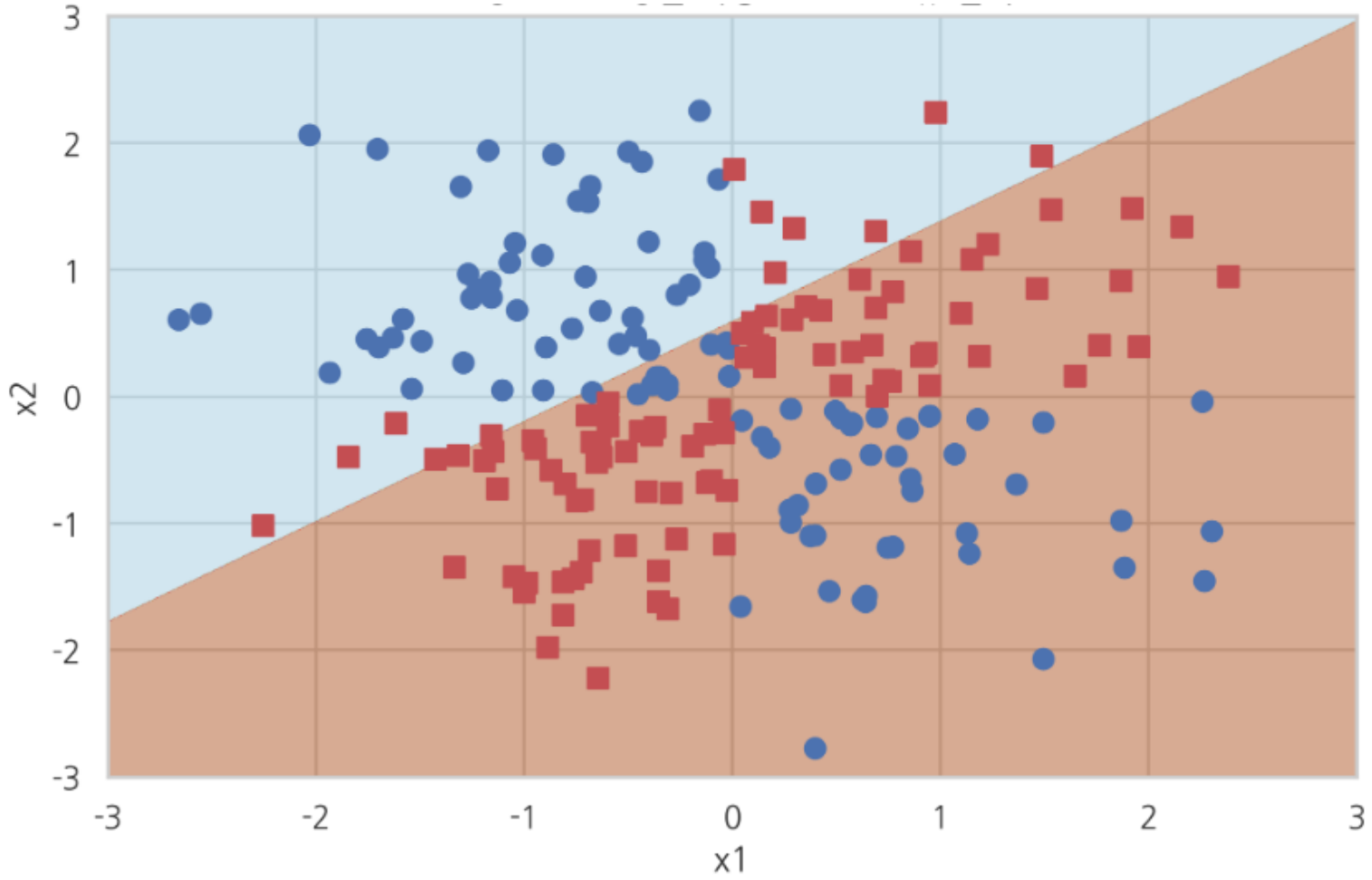
$$X = (x_1, x_2, x_3, \dots) \rightarrow \phi(x) = Z = (Z_1, \dots)$$



Kernel Function

Linear Kernel 

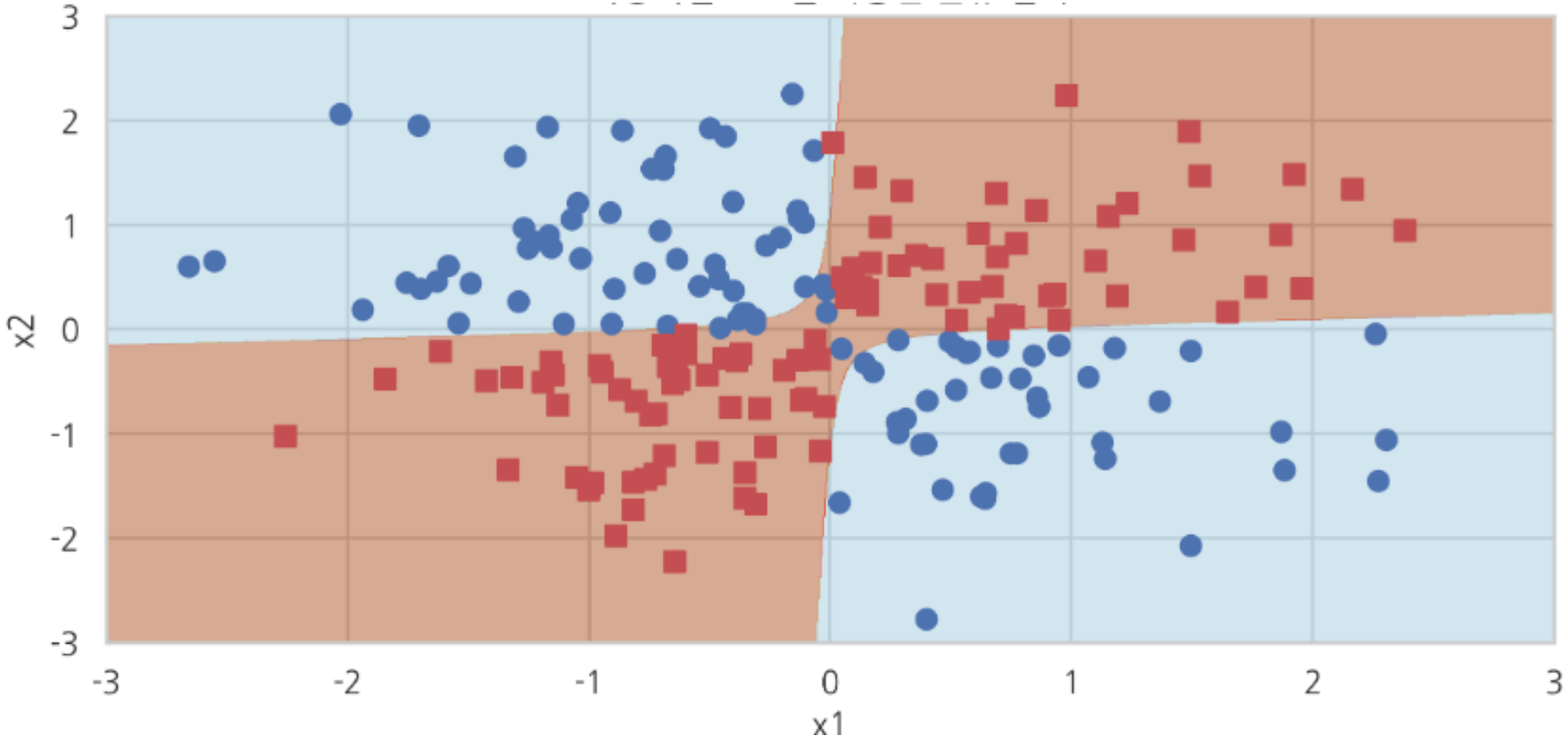
$$K(x_1, x_2) = (x_1, x_2)$$



Kernel Function

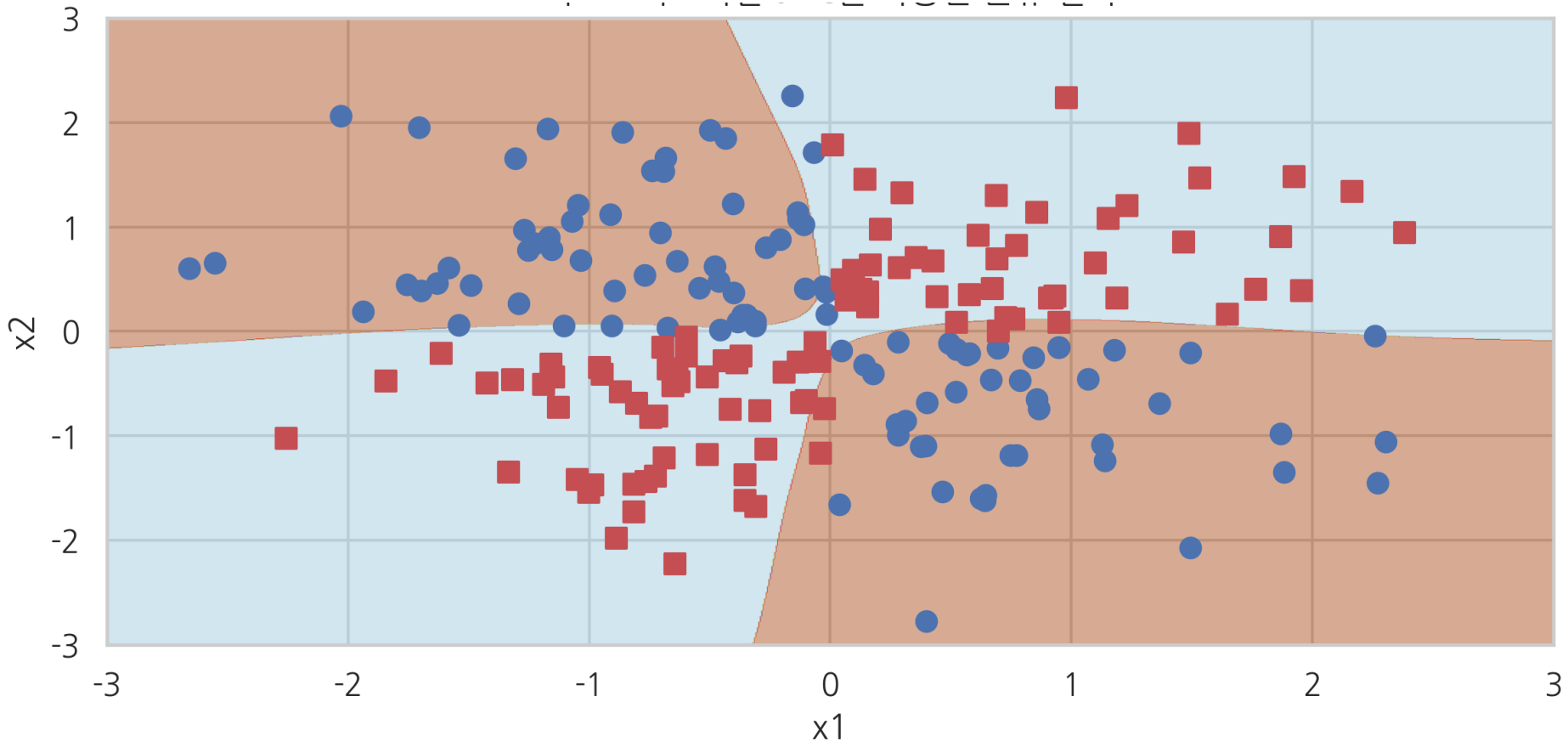
Polynomial Kernel

$$K(x_1, x_2) = (a(x_1, x_2) + b)^d$$



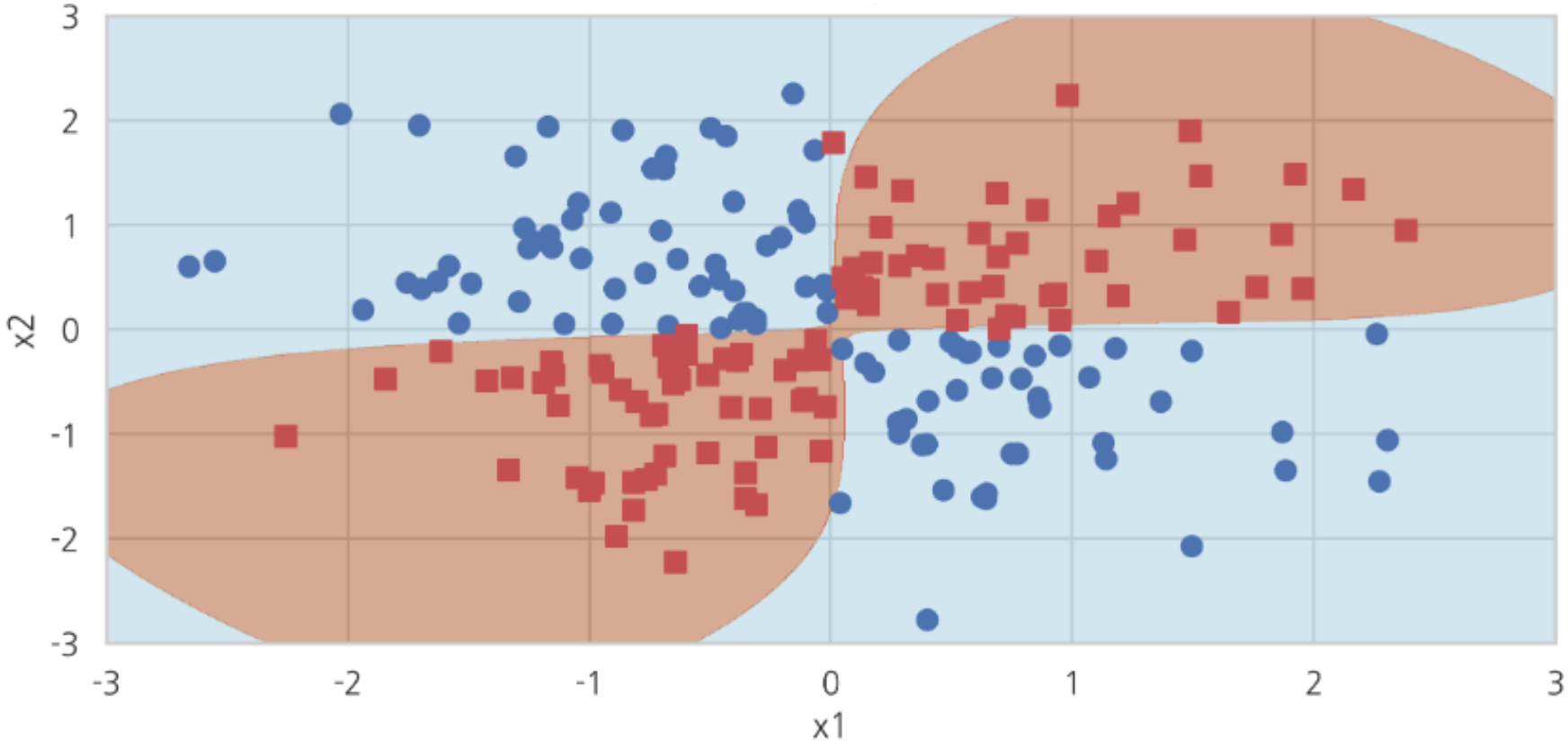
Sigmoid Kernel

$$K(x_1, x_2) = tanh(a(x_1, x_2) + b)$$



Radial Kernel 

$$K(x_1, x_2) = e^{\frac{-||x_1 - x_2||^2}{2\sigma^2}}$$



Kernel SVM

 Kernel Function의 결정 기준이 없다! 

데이터 특성에 맞는 Kernel을 결정하자!

01

임공주

하~ 정말 완벽히 설명했어
그럼 실습으로 가보자 친구들^_^

실습해보자1 

e1071을 사용한 SVM

iris_data: 꽃받침의 길이와 폭, 꽃잎의 길이와 폭에 대한 값이다.

꽃의 종류로는 setosa, versicolor, virginica가 있다. 총 150개의 관측치가 있다.

Class 변수인 Species가 이미 factor이므로 바로 분석을 시작하자!

```
> # install.packages("e1071")
> library("e1071")
> str(iris)
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ..
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ..
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1
```

실습해보자1 

e1071을 사용한 SVM

```
> # iris 데이터를 train과 test로 나눔
> library(caret)
> set.seed(100)
> train_index = createDataPartition(y = iris$Species, p=0.7, list=FALSE)
> train = iris[train_index,]
> test = iris[-train_index,]
> # 간단하게 SVM 모델 만들기
> svm_model = svm(Species~., data=train)
> summary(svm_model)

Call:
svm(formula = Species ~ ., data = train)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  43

 ( 7 18 18 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

1. 데이터를 train과 test로 나눈다.
2. 간단하게 svm model을 돌려본다.

<Default Model>

svm(formula, data = NULL, ..., subset,
na.action =na.omit, scale = TRUE)

실습해보자1 

e1071을 사용한 SVM

SVM() 함수의 옵션들

- Type : svm()의 수행 방법(분류, 회귀 또는 novelty detection)을 정한다. 반응변수 (y)가 범주형인지의 여부에 따라 정해지며, 디폴트는 C-classification 또는 eps-regression이다. 외에도 nu-classification, one-classification(for novelty detection), nu-regression이 있다.
- Kernel : 훈련과 예측에 사용되는 커널로, “radial” 옵션은 가우시안 RBF를 의미한다. 커널에 맞는 parameter도 고려해야한다.
- degree : 다항커널(polynomial)이 사용될 경우의 모수(차수)로, 디폴트는 3이다.
- Gamma : 선형(linear)을 제외한 모든 커널에 요구되는 모수로, 디폴트는 1/(데이터차원) 이다.
- coef0 : 다항(polynomial) 또는 시그모이드(sigmoid) 커널에 요구되는 모수로, 디폴트는 0이다.
- cost : 제약 위배의 비용으로, 디폴트는 1이다.
- cross : k-중첩 교차타당도에서 k값을 지정한다.

옵션에 대하여 더 궁금하다면 여기를 참고하자!
⇒ https://www.rdocumentation.org/packages/e1071/versions/1.7-3/topics/svm

실습해보자1 

e1071을 사용한 SVM

```
> # predict 해보기
> pred = predict(svm_model,test)
> confusionMatrix(pred, test$Species)
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         14          0         0
  versicolor      1         13         0
  virginica       0          2        15

Overall Statistics

               Accuracy : 0.9333
                 95% CI : (0.8173, 0.986)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 0.9333            0.8667           1.0000
Specificity                 1.0000            0.9667           0.9333
Pos Pred Value              1.0000            0.9286           0.8824
Neg Pred Value              0.9677            0.9355           1.0000
Prevalence                  0.3333            0.3333           0.3333
Detection Rate              0.3111            0.2889           0.3333
Detection Prevalence        0.3111            0.3111           0.3778
Balanced Accuracy           0.9667            0.9167           0.9667
```

3. 아까 만든 모델에 대해 얼마나 예측했나 보자!

정확도는 93%정도이다!

튜닝을 하지 않아도 나쁘지 않은 결과를 보여주네!

그래도 튜닝을 해보자~~^0^

실습해보자1 

e1071을 사용한 SVM

```
> # kernel에 따른 최적의 parameter 찾기
> tune1 = tune.svm(Species~. ,data=train, gamma=10^(-5:0),cost = 2^(0:4), kernel="radial")
> tune2 = tune.svm(Species~., data=train, cost=2^(0:4), kernel="linear")
> tune3 = tune.svm(Species~., data=train, cost=2^(0:4), degree=2:4, kernel = "polynomial")
```

4. 여러 커널별로 최적의 parameter를 찾아보자! (tune.svm())

```
tune1 # gamma 0.1 / cost 2
tune2 # cost 1
tune3 # degree 3 / cost 8
```

```
> tune1 # gamma 0.1 / cost 2

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
   0.1    8

- best performance: 0.03818182
```

```
> tune2 # cost 1

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
   16

- best performance: 0.01909091
```

```
> tune3 # degree 3 / cost 8

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 degree cost
      3    8

- best performance: 0.05636364
```

실습해보자1 

e1071을 사용한 SVM

5. 찾은 최적값들로 모델을 만들어보자! (radial / linear)

```
> # 찾은 최적값으로 모델 다시 만들기
> model_1 = svm(Species~., data=train, kernel="radial", cost=2, gamma=0.1)
> summary(model_1)

Call:
svm(formula = Species ~ ., data = train, kernel = "radial", cost = 2, gamma = 0.1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  2

Number of Support Vectors:  36

 ( 3 17 16 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

```
> model_2 = svm(Species~., data=train, kernel="linear", cost=1)
> summary(model_2)

Call:
svm(formula = Species ~ ., data = train, kernel = "linear", cost = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  25

 ( 2 12 11 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

실습해보자1 

e1071을 사용한 SVM

5. 찾은 최적값들로 모델을 만들어보자! (polynomia)

```
> model_3 = svm(Species~., data=train, kernel="polynomia", cost=8, degree=3)
> summary(model_3)

Call:
svm(formula = Species ~ ., data = train, kernel = "polynomia", cost = 8,
    degree = 3)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  8
     degree:  3
     coef.0:  0

Number of Support Vectors:  26

 ( 1 13 12 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

```
> # 서포트벡터 확인(몇 번째 관찰값이 서포트 벡터일까!)
> model_1$index
 [1]  10  17  32  36  37  38  39  40  43  45  46  47
[22]  75  79  81  82  83  87  89  90  93  94  95  98
> model_2$index
 [1]  17  32  36  37  39  44  46  47  49  55  56  57
[22]  95  98  99 105
> model_3$index
 [1]  17  37  38  39  40  42  44  47  49  55  58  61
[22]  94  95  98  99 105
> # predict 해보기
> pred_1 = predict(model_1, test)
> pred_2 = predict(model_2, test)
> pred_3 = predict(model_3, test)
```

6. 각 모델들의 서포트벡터 확인 및 predict하기

실습해보자1 

e1071을 사용한 SVM

```
> confusionMatrix(pred_1, test$Species)
Confusion Matrix and Statistics

            Reference
Prediction    setosa versicolor virginica
  setosa          15          0         0
  versicolor       0         13         0
  virginica        0          2        15

Overall Statistics

               Accuracy : 0.9556
                 95% CI : (0.8485, 0.9946)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9333

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            0.8667           1.0000
Specificity                 1.0000            1.0000           0.9333
Pos Pred Value              1.0000            1.0000           0.8824
Neg Pred Value              1.0000            0.9375           1.0000
Prevalence                  0.3333            0.3333           0.3333
Detection Rate              0.3333            0.2889           0.3333
Detection Prevalence        0.3333            0.2889           0.3778
Balanced Accuracy           1.0000            0.9333           0.9667
```

```
> confusionMatrix(pred_2, test$Species)
Confusion Matrix and Statistics

            Reference
Prediction    setosa versicolor virginica
  setosa          15          0         0
  versicolor       0         13         0
  virginica        0          2        15

Overall Statistics

               Accuracy : 0.9556
                 95% CI : (0.8485, 0.9946)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9333

                  ue : NA

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            0.8667           1.0000
Specificity                 1.0000            1.0000           0.9333
Pos Pred Value              1.0000            1.0000           0.8824
Neg Pred Value              1.0000            0.9375           1.0000
Prevalence                  0.3333            0.3333           0.3333
Detection Rate              0.3333            0.2889           0.3333
Detection Prevalence        0.3333            0.2889           0.3778
Balanced Accuracy           1.0000            0.9333           0.9667
```

7. Confusion Matrix를 통해 정확도를 보자!

실습해보자1 

e1071을 사용한 SVM

```
> confusionMatrix(pred_3, test$Species)
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         15          0         0
  versicolor      0         14         0
  virginica       0          1        15

Overall Statistics

               Accuracy : 0.9778
                 95% CI : (0.8823, 0.9994)
    No Information Rate : 0.3333
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9667

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            0.9333           1.0000
Specificity                 1.0000            1.0000           0.9667
Pos Pred Value              1.0000            1.0000           0.9375
Neg Pred Value              1.0000            0.9677           1.0000
Prevalence                  0.3333            0.3333           0.3333
Detection Rate              0.3333            0.3111           0.3333
Detection Prevalence        0.3333            0.3111           0.3556
Balanced Accuracy           1.0000            0.9667           0.9833
```

8. Model3가 정확도가 97.8%로 제일 높다.

따라서 model3로 분류를 시각화해보자!

(튜닝하기 전에는 93.3%였는데 튜닝 이후에 모델들 정확도가 다 향상된 것을 볼 수 있다!)

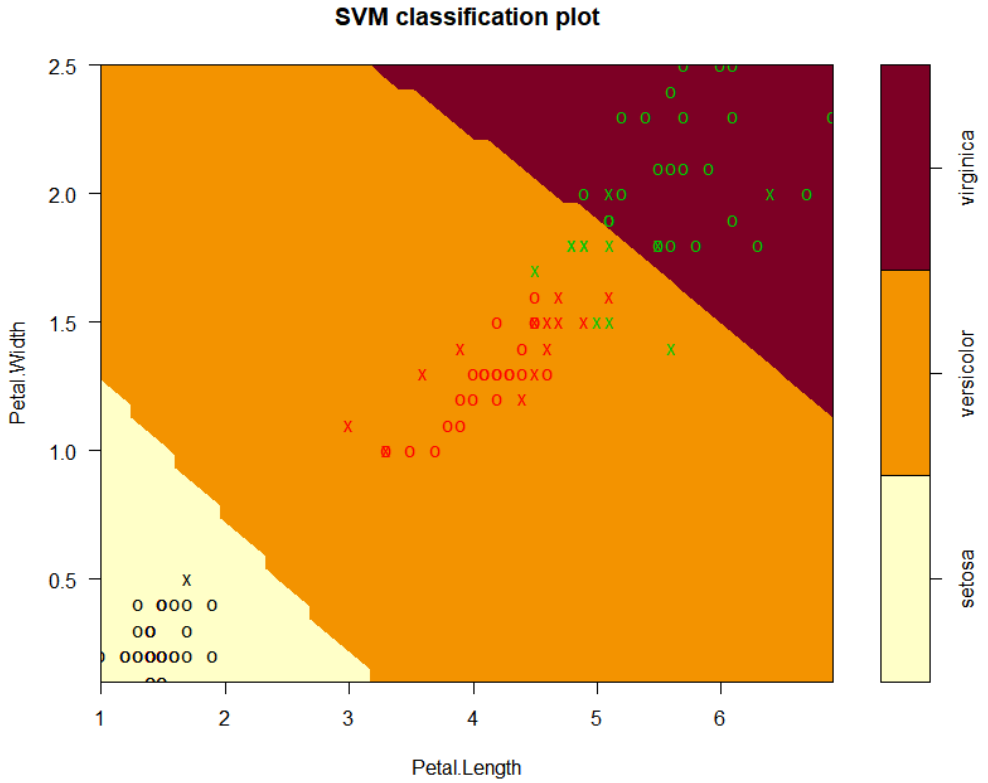
```
# 시각화
plot(model_3, train, Petal.Width~ Petal.Length,
     slice = list(Sepal.Width=3, Sepal.Length=5))

plot(model_3, train, Sepal.Width~ Sepal.Length,
     slice = list(Petal.Width=2.5, Petal.Length=3))
```

실습해보자1 

e1071을 사용한 SVM

* slice = 시각화를 위해 두 변수만 쓰기 때문에 나머지 변수에는 값을 주기 위해서 하는 것!

```
> # 시각화
> plot(model_3, train, Petal.Width~ Petal.Length,
+      slice = list(Sepal.Width=3, Sepal.Length=5))
```



X = 서포트벡터

O = 데이터

```
> plot(model_3, train, Sepal.Width~ Sepal.Length,
+      slice = list(Petal.Width=2.5, Petal.Length=3))
```

실습해보자2 

Kernlab을 사용한 SVM

letterdata.csv : 문자부류(letter class) 예시를 정의하는 16개의 특징으로 이뤄진 데이터

예상대로 문자는 26개의 레벨을 갖는다! A~Z

```
> letters <- read.csv("letterdata.csv")
> str(letters)
'data.frame':   20000 obs. of  17 variables:
 $ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1
 10 13 ...
 $ xbox  : int  2 5 4 7 2 4 4 1 2 11 ...
 $ ybox  : int  8 12 11 11 1 11 2 1 2 15 ...
 $ width : int  3 3 6 6 3 5 5 3 4 13 ...
 $ height: int  5 7 8 6 1 8 4 2 4 9 ...
 $ onpix : int  1 2 6 3 1 3 4 1 2 7 ...
 $ xbar  : int  8 10 10 5 8 8 8 8 10 13 ...
 $ ybar  : int  13 5 6 9 6 8 7 2 6 2 ...
 $ x2bar : int  0 5 2 4 6 6 6 2 2 6 ...
 $ y2bar : int  6 4 6 6 6 9 6 2 6 2 ...
 $ xybar : int  6 13 10 4 6 5 7 8 12 12 ...
 $ x2ybar: int  10 3 3 4 5 6 6 2 4 1 ...
 $ xy2bar: int  8 9 7 10 9 6 6 8 8 9 ...
 $ xedge : int  0 2 3 6 1 0 2 1 1 8 ...
 $ xedgey: int  8 8 7 10 7 8 8 6 6 1 ...
 $ yedge : int  0 4 3 2 5 9 7 2 1 1 ...
 $ yedgex: int  8 10 9 8 10 7 10 7 7 8 ...
```

- SVM 학습자는 모든 특징이 수치여야함
- 모든 특징이 정수이므로 팩터를 숫자로 변환할 필요 없다!
- 정수 변수의 일부 범위가 상당히 넓게 나타난다.
- 데이터를 정규화하거나 표준화할 필요가 있음을 의미하지만 모델을 적합시키기 위해 사용할 R패키지가 자동으로 재조정을 실행해줌

02

실습해보자2 

Kernlab을 사용한 SVM

이미 데이터가 랜덤화되어 있는 데이터이므로 처음 16,000레코드(80%)를 모델 구축에 사용하고,

다음 4,000개 레코드(20%)를 테스트에 사용

```
> letters_train <- letters[1:16000, ]
> letters_test  <- letters[16001:20000, ]
```

분류기 구축 : kernlab 패키지의 ksvm( ) 함수 사용

```
m = ksvm(target ~ predictors, data = mydata,
         kernel = "rbfdot", c = 1)
```

- Target은 모델링될 mydata 데이터 프레임 내 출력
- Predictors는 모델에서 사용할 mydata 데이터 프레임 내 특징을 명시하는 R 구문
- Data는 target과 predictors변수를 찾을 수 있는 데이터 프레임
- Kernel은 "rbfoot"(방사형기저), "polydot"(다항), "tanhdot"(하이퍼볼릭 탄젠트 시그모이드), "vanilladot"(선형)과 같은 비선형 매핑
- C는 제약을 위반할 때의 비용, 즉 '소프트 마진 ' 에 대해 패널티가 얼마나 큰지를 지정하는 숫자로, 이 값이 커질수록 여백은 좁아진다.

실습해보자2 

Kernlab을 사용한 SVM

예측: p = predict(m, test, type="response")

- m은 ksvm( ) 함수에 의해 훈련된 모델
- Test는 분류기를 구축하는 데 사용된 훈련 데이터와 같은 특징을 갖는 테스트 데이터를 포함하는 데이터 프레임
- Type은 예측이 "response"(예측 클래스)인지 "probabilities"(예측 확률, 클래스 레벨별로 하나의 열)인지를 지정
- 이 함수는 type 파라미터의 값에 따라 예측 클래스(혹은 확률)의 벡터(또는 행렬)을 반환

```
> #install.packages("kernlab")
> library(kernlab)
> letter_classifier <- ksvm(letter ~ ., data = letters_train,
+                           kernel = "vanilladot")
 Setting default kernel parameters
> # look at basic information about the model
> letter_classifier
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 7037

Objective Function Value : -14.1746 -20.0072 -23.5628 -6.2009 -7
1.2195 -13.276 -35.6217 -30.8612 -16.5256 -14.6811 -32.7475 -30.
Training error : 0.130062
```

선형 커널로 훈련데이터에 대해 ksvm( )함수를 호출하자

학습한 모델의 정보를 보면 실제 얼마나 잘 실행될 것인지에 대해서는 거의 알려주지 않는다
모델이 처음 보는 데이터에 대해 일반화를 잘하는지 알려면 테스트 데이터셋에 대한 성능 검토를 해보자

실습해보자2 

Kernlab을 사용한 SVM

```
> letter_predictions <- predict(letter_classifier, letters_test)
> head(letter_predictions)
[1] U N V X N H
Levels: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
> table(letter_predictions, letters_test$letter)

letter_predictions   A   B   C   D   E   F   G   H   I   J   K
                 A 144   0   0   0   0   0   0   0   0   1   0
                 B   0 121   0   5   2   0   1   2   0   0   1
                 C   0   0 120   0   4   0  10   2   2   0   1
                 D   2   2   0 156   0   1   3  10   4   3   4
                 E   0   0   5   0 127   3   1   1   0   0   3
                 F   0   0   0   0   0 138   2   2   6   0   0
                 G   1   1   2   1   9   2 123   2   0   0   1
                 H   0   0   0   1   0   1   0 102   0   2   3
                 I   0   1   0   0   0   1   0   0 141   8   0
                 J   0   1   0   0   0   1   0   2   5 128   0
                 K   1   1   9   0   0   0   2   5   0   0 118
```

predict( ) 함수를 이용하여 테스트 데이터셋 예측
-> 여기서 type 파라미터를 지정하지 않았기 때문에 type = “response” 디폴트가 사용됐다.

이 함수는 테스트 데이터의 각 행에 대해 예측된 문자가 포함된 벡터를 반환

분류기가 얼마나 잘 실행됐는지 검토하기 위해 예측된 문자와 테스트 데이터셋에 있는 실제 문자를 비교해보자

table( ) 함수를 통해 확인!

전체적인 정확도 계산을 통해 평가를 간소화해보자!
이는 예측이 옳은지 또는 그른지에 대해서만 고려하고 오류 유형은 무시한다.

실습해보자2 

Kernlab을 사용한 SVM

```
> agreement <- letter_predictions == letters_test$letter
```

TRUE 또는 FALSE 값의 벡터를 반환해줌

-> 모델의 예측된 문자가 테스트 데이터셋에 있는 실제 문자와 일치하는지를 나타내줌

```
> table(agreement)
agreement
FALSE  TRUE
  643  3357
```

Table( ) 함수로 분류기가 테스트 레코드 4,000개 중 3,357개의 문자를 정확히 식별했다는 것을 알 수 있음

```
> prop.table(table(agreement))
agreement
  FALSE    TRUE
0.16075 0.83925
```

백분율로 정확도는 약 84%이다.

지금까지는 단순한 선형 커널 함수를 사용했다면, 좀 더 복잡한 커널 함수를 통해 데이터를 더 높은 차원의 공간으로 매핑해보자!

실습해보자2 

Kernlab을 사용한 SVM

일반적으로 관형 중 하나는 가우시안 RBF를 많이 사용한다고 하니 우리도 RBF 기반의 SVM을 훈련해보자

```
> set.seed(12345)
> letter_classifier_rbf <- ksvm(letter ~ ., data = letters_train, kernel = "rbfdot")
> letter_predictions_rbf <- predict(letter_classifier_rbf, letters_test)
>
> agreement_rbf <- letter_predictions_rbf == letters_test$letter
> table(agreement_rbf)
agreement_rbf
FALSE  TRUE
  275  3725
> prop.table(table(agreement_rbf))
agreement_rbf
  FALSE    TRUE
0.06875 0.93125
```

단순히 커널 함수를 바꿈으로써 문자 인식 모델의 정확도를 84%에서 93%로 올릴 수 있었다!

이 외에도 앞에 설명이 있었던 다른 커널을 통해 테스트하거나 결정 경계의 폭을 바꾸기 위해 제약의 비용 파라미터 C에 변화를 줄 수도 있다!