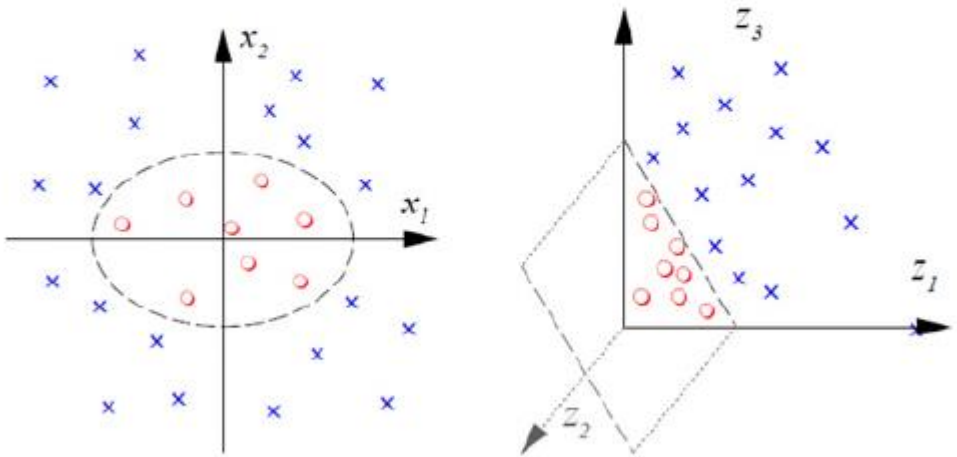
데이터마이닝
(**D**ata **M**ining)

Chapter 2.2 Support Vector Machine

- SVM은 비선형 결정 경계(non-linear decision boundaries)를 가지는 문제도 해결할 수 있다.
- 핵심 아이디어는 원래의 d-차원 공간을 d'-차원($d' > d$)으로 맵핑하여 그 점들이 선형적으로 분리 가능하도록 하는 것이다. 원 자료집합 $D = \{(x_i, y_i), i = 1,2, \dots, n\}$와 변환함수 Φ가 주어질 때, 새로운 자료집합은 변환공간에서 $D_\Phi = \{(\Phi(x_i), y_i), i = 1,2, \dots, n\}$으로 구해진다.

- 다항 맵핑을 사용한 예를 보여준다.

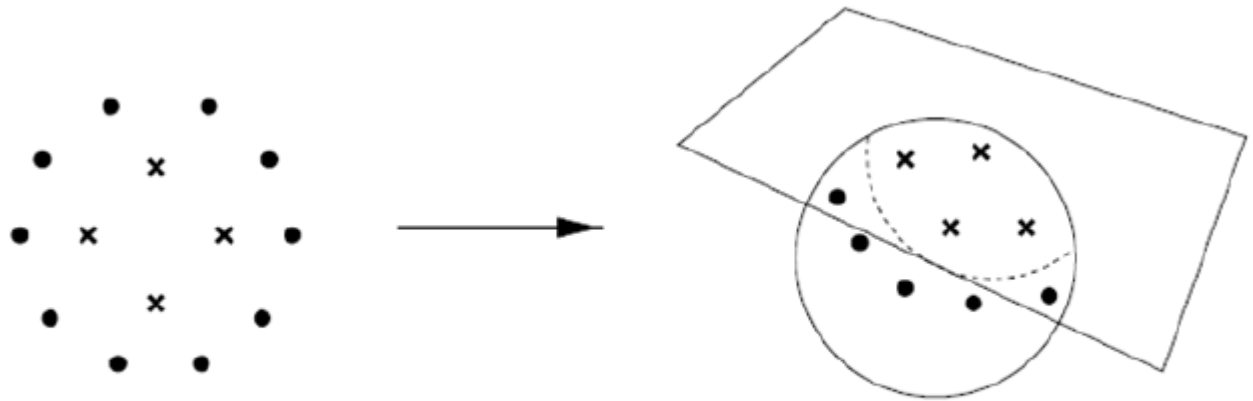
$$\Phi : R^2 \rightarrow R^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)}x_1x_2, x_2^2)$$



- 이제 선형결정면(linear decision surface)을 d'-차원에서 구한 후, 이를 다시 원래의 d-차원 공간상의 비선형 면으로 맵핑한다.
- 이 때, 변환된 공간(d'-차원) 에서의 선형결정면은

$$f(x) = w \cdot \Phi(x) + b$$

이며, 이 값의 부호를 통해 분류를 수행하게 된다



(a) 맵핑 전

(b) 맵핑 후

- 이 과정에서 w와 b를 구하기 위해 맵핑된 결과 $\Phi(x)$는 별도로 계산될 필요는 없다. 그 이유는 w가 다음과 같이

$$w = \sum_{i=1}^{n} \alpha_i \Phi(x_i) \quad \text{for some variables } \alpha$$

으로 표현될 수 있음이 밝혀져 있어(Representer 정리, Kimeldorf & Wahba(1971)), w를 직접 최적화하는 대신 α를 최적화 할 수 있기 때문이다.

- 이 때, 판별규칙은

$$f(x) = \sum_{i=1}^{n} \alpha_i \, \Phi(x_i) \cdot \Phi(x) + b$$

이 되며, $K(x_i, x) = \Phi(x_i) \cdot \Phi(x)$를 커널함수(kernel function)라 부른다.

- 따라서 변환된 공간에서 요구되는 유일한 연산은 내적(inner product) $\Phi(x_i)^T \Phi(x)$으로, 이는 x_i와 x간의 커널함수(K)와 함께 정의된다.
- 실제 Φ의 선택에 따라 맵핑된 공간의 차원이 매우(무한 차원까지) 커질 수 있어 Φ를 직접 이용하거나 맵핑된 자료의 내적을 계산하는데 어려움이 있다.

- 이를 극복하는 방법은 다음과 같다. 두 점 p와 q에 대해 맵핑 $\Phi: (p_1, p_2) \rightarrow (p_1^2, \sqrt{2}p_1p_2, p_2^2)$를 적용한 후, 그들의 내적을 계산하면 다음과 같다.
- $$\begin{aligned}\Phi(p) \cdot \Phi(q) &= (p_1^2, \sqrt{2}p_1p_2, p_2^2)\left(p_1^2, \sqrt{2}p_1p_2, p_2^2\right)^{\mathrm{T}} \\ &= p_1^2q_1^2 + 2p_1q_1p_2q_2 + p_2^2q_2^2 \\ &= (p_1q_1 + p_2q_2)^2 \\ &= (p \cdot q)^2.\end{aligned}$$
- 위 결과로부터 맵핑된 점들 간의 내적은 원 자료의 내적을 계산한 뒤 제곱을 취해 구할 수 있다. 따라서 Φ함수의 적용 없이 내적 $\Phi(p) \cdot \Phi(q)$을 계산할 수 있다. 맵핑 공간에서의 내적과 동등한 함수(equivalent function)를 커널함수라 하고 이를 K라 표현한다.

- SVM에서 통상적으로 사용되는 커널은 다음과 같다.
  - 선형(linear) 커널 :

$$K(x_i, x_j) = x_i^T x_j$$

  - 다항(polynomial) 커널:

$$K(x_i, x_j) = (x_i^T x_j + 1)^q, \quad q: \text{다항식의 차수}$$

  - 가우시안(gaussian) 커널:

$$K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}, \quad \sigma: \text{퍼진정도(spread) 또는 표준편차}$$

- 가우시안 RBF(gaussian radial basis function) 커널 :

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \qquad \gamma \geq 0$$

- 라플라스 RBF(Laplace radial basis function) 커널:

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|}, \qquad \gamma \geq 0$$

- 역탄젠트(hyperbolic tangent) 커널:

$$K(x_i, x_j) = tanh(x_i^T x_j + offset)$$

- 시그모이드(sigmoid) 커널 :

$$K(x_i, x_j) = tanh(ax_i^T x_j + offset)$$

- 제1종 베셀함수(the Bessel function of the first kind) 커널:

$$K(x_i, x_j) = \frac{Bessel_{v+1}^{n}(\sigma\|x_i - x_j\|)}{(\|x_i - x_j\|)^{-n(v+1)}}$$

- ANOVA Radial basis 커널 :

$$K(x_i, x_j) = \left(\sum_{k=1}^{n} e^{-\sigma\left(x_i^k - x_j^k\right)^2} \right)^d$$

- 일차원의 선형스플라인(linear splines kernel in one dimension) 커널 :

$$K(x_i, x_j) = 1 + x_i x_j \min(x_i, x_j) - \frac{(x_i + x_j)}{2} \min(x_i, x_j)^2 + \frac{\min(x_i, x_j)^3}{3}$$

- Karatzoglou 등(2006)에 의하면, 가우시안과 라플라스 RBF, 베셀 커널은 자료에 관한 사전정보가 없을 때 사용되는 일반적-목적의 커널이다.
- 선형커널은 문서분류(text categorization) 등에서 자주 발생하는 대량의 희박 자료벡터를 다룰 때 유용하다.
- 다항커널은 이미지 처리에 자주 사용되며, 시그모이드 커널은 신경망에 대한 프록시(proxy)로 주로 사용된다. 스플라인과 ANOVA RBF 커널은 전형적으로 회귀 문제에 잘 수행된다.
- SVM을 수행하는 R 패키지에는 {e1071}, {kernlab}, {klaR}, {svmpath}, {shogun} 등이 있다. 이 가운데 {e1071} 패키지가 R에서 가장 먼저 소개되었으며 가장 직관적이다.

- 다음의 [예제 1]은 R 패키지 {e1071}의 svm() 함수를 이용하여 SVM 분류를 수행한다.

예제 1 iris 자료를 이용하여 SVM을 수행한다.

```
> library("e1071")
> data(iris)
```

- svm() 함수를 통해 분류를 수행한다. svm() 함수에는 사용되는 중요 옵션은 다음과 같다.
  - type : svm()의 수행 방법(분류, 회귀 또는 novelty detection)을 정한다. 반응변수(y) 가 범주형인지의 여부에 따라 정해지며, 디폴트는 C-classification 또는 eps-regression 이다. rm 외에도 nu-classification, one-classification(for novelty detection), nu-regression이 있다.
  - kernel : 훈련과 예측에 사용되는 커널로, "radial" 옵션은 가우시안 RBF를 의미한다. 실제 문제에서 커널의 선택이 결과의 정확도에 큰 영향을 주지는 않는다.

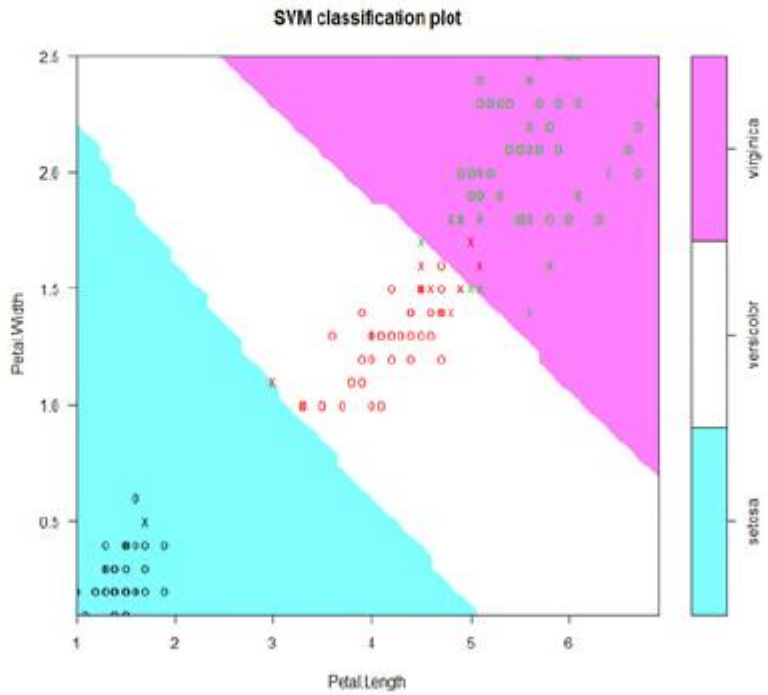
- degree : 다항커널이 사용될 경우의 모수(차수)이다.
- gamma : 선형을 제외한 모든 커널에 요구되는 모수로, 디폴트는 1/(데이터 차원) 이다.
- coef0 : 다항 또는 시그모이드 커널에 요구되는 모수로, 디폴트는 0 이다.
- cost : 제약 위배의 비용으로, 디폴트는 1 이다.
- cross : k- 중첩 교차타당도에서 k값을 지정한다.

```
> svm.e1071 <- svm(Species ~ . , data = iris,
                   type = "C-classification", kernel = "radial",
                   cost = 10, gamma = 0.1)
> summary(svm.e1071)

                                (…)
```

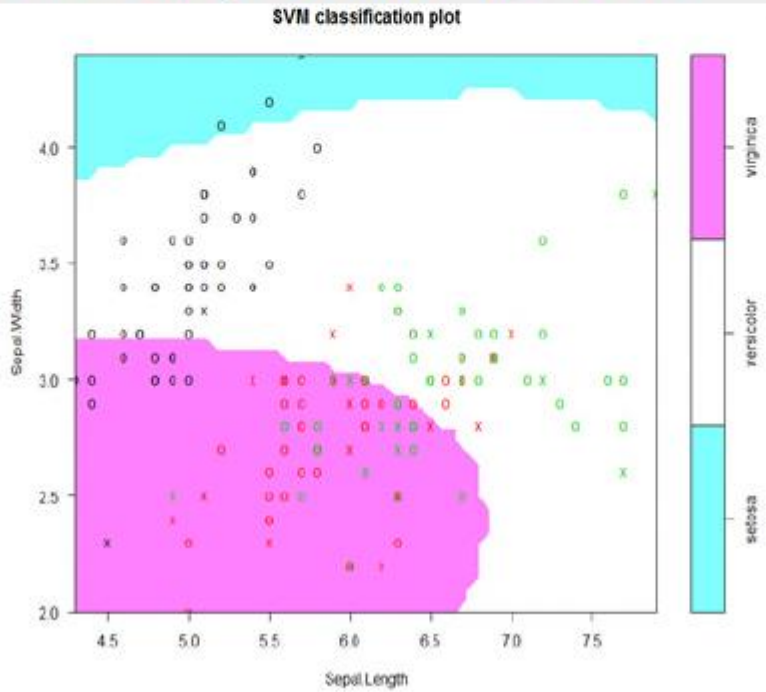
```
Call:
 svm(formula = Species ~ ., data = iris, type = "C-classification",
         kernel = "radial", cost = 10, gamma = 0.1)

 Parameters:
    SVM-Type: C-classification
  SVM-Kernel: radial
        cost: 10
       gamma: 0.1

 Number of Support Vectors: 32
   ( 3 16 13 )

 Number of Classes: 3

 Levels:
  setosa versicolor virginica
```

- plot() 함수를 이용하여 그 결과에 대한 시각화 할 수 있다.

```
> plot(svm.e1071, iris, Petal.Width ~ Petal.Length,
       slice = list(Sepal.Width = 3, Sepal.Length = 4))
```



```
> plot(svm.e1071, iris, Sepal.Width ~ Sepal.Length,
       slice = list(Petal.Width = 2.5, Petal.Length = 3))
> # slice= 변수가 2개 이상일 때 상수값을 할당함(assign)
> # 아래 그림에서 x: 서포트벡터, o:데이터 점을 나타냄
```



```
> pred <- predict(svm.e1071, iris, decision.values = TRUE)
> (acc <- table(pred, iris$Species))
pred          setosa versicolor virginica
  setosa          50          0         0
  versicolor       0         47         0
  virginica        0          3        50
```

- 분류된 데이터를 실제 값과 비교해보면 setosa는 50개 모두 잘 분류 되었고, versicolor은 50개 중 47개가 잘 분류되었으며 virginica는 50개 모두 잘 분류되었다.

- classAgreement() 함수를 통해 모형의 정확도를 확인할 수 있다.

```
> classAgreement(acc)
$diag
[1] 0.98

$kappa
[1] 0.97

$rand
[1] 0.9739597

$crand
[1] 0.941045
```

- tune() 함수는 제공된 모수 영역에서 격자 탐색을 사용하여 통계적 방법의 초모수 (hyperparameters)를 조절(tune)할 수 있다. 이 함수는 최적의 모수를 제공해 주며, 동시에 여러 모수 값에 대한 검정에 대한 자세한 결과를 제공해 준다.

```
> tuned <- tune.svm(Species~., data = iris, gamma = 10^(-6:-1),
                    cost = 10^(1:2))
> # 6× 2 = 12개의 조합에서 모수조율이 이루어짐
> summary(tuned)

 Parameter tuning of ‘svm’:
 - sampling method: 10-fold cross validation
 - best parameters:
 gamma cost
  0.01  100
                                        (…)
```

```
- best performance: 0.03333333
- Detailed performance results:
   gamma cost      error dispersion
1  1e-06   10 0.78666667 0.07568616
2  1e-05   10 0.78666667 0.07568616
3  1e-04   10 0.63333333 0.15153535
4  1e-03   10 0.10666667 0.10976968
5  1e-02   10 0.04000000 0.04661373
6  1e-01   10 0.04000000 0.04661373
7  1e-06  100 0.78666667 0.07568616
8  1e-05  100 0.63333333 0.15153535
9  1e-04  100 0.10000000 0.10999439
10 1e-03  100 0.04000000 0.04661373
11 1e-02  100 0.03333333 0.04714045
12 1e-01  100 0.04666667 0.04499657
```

- 다음의 [예제 2]는 R 패키지 {kernlab}의 ksvm() 함수를 이용하여 SVM 분류를 수행한다.

예제 2 iris 자료를 이용하여 SVM을 수행한다.

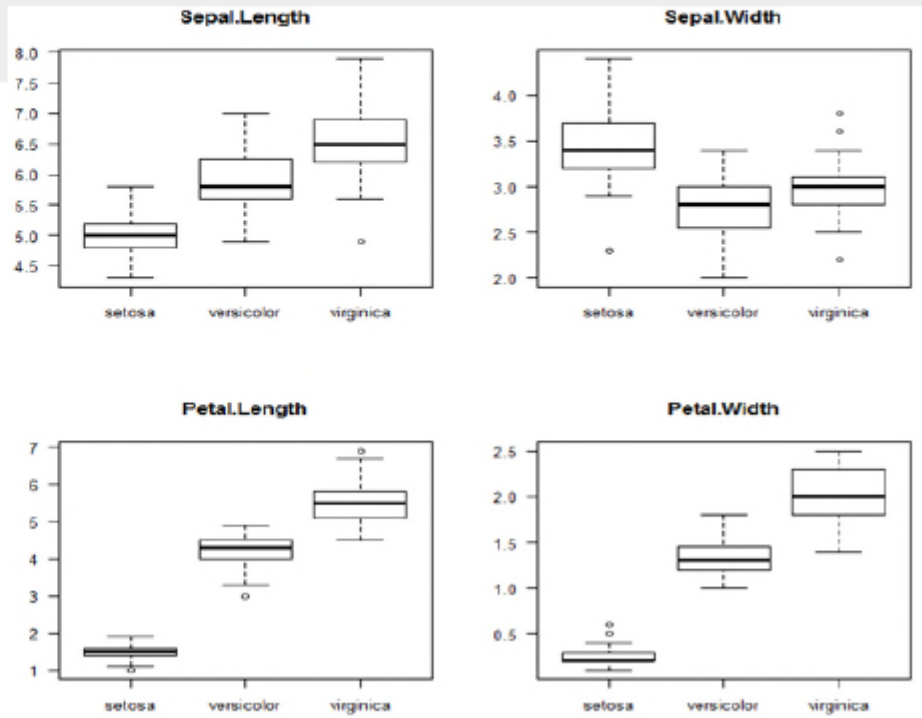
```
> library("kernlab")
> data(iris)
> svm.kernlab <- ksvm(Species ~ ., data = iris,  type = "C-bsvc",
                      kernel = "rbfdot", kpar = list(sigma = 0.1),
                      C = 10, prob.model = TRUE)
> svm.kernlab
Support Vector Machine object of class "ksvm"

SV type: C-bsvc (classification)
parameter : cost C = 10
                                  (…)
```

```
Gaussian Radial Basis kernel function.
   Hyperparameter : sigma = 0.1

 Number of Support Vectors : 32

 Objective Function Value : -5.8442 -3.0652 -136.9786
 Training error : 0.02
 Probability model included.
```

- plot() 함수를 이용하여 분류된 결과에 대한 각 변수별 분포를 상자그림의 형태로 나타낼 수 있다.

```
> fit <- fitted(svm.kernlab)
> par(mfrow=c(2,2))
> plot(fit, iris[,1], main="Sepal.Length")
                                    (…)
```

```
> plot(fit, iris[,2], main="Sepal.Width")
> plot(fit, iris[,3], main="Petal.Length")
> plot(fit, iris[,4], main="Petal.Width")
> par(mfrow=c(1,1))
```



```
> head(predict(svm.kernlab, iris, type= "probabilities"))
> # type= “probabilities”, “decision”, “response”, “votes”
          setosa   versicolor   virginica
 [1,] 0.982512676 0.0107155952 0.006771729
 [2,] 0.976761862 0.0155766815 0.007661456
 [3,] 0.984482548 0.0088730948 0.006644357
 [4,] 0.980694369 0.0116272964 0.007678335
 [5,] 0.983721897 0.0096395760 0.006638527
 [6,] 0.969339542 0.0213361722 0.009324286

> head(predict(svm.kernlab, iris, type = "decision"))
> # 3개의 이진분류기의 decision value
           [,1]       [,2]        [,3]
 [1,] -1.3982580 -1.1850725 -4.22358404
 [2,] -1.2750649 -1.1486247 -4.22523983
 [3,] -1.4603977 -1.1910251 -3.88688364
 [4,] -1.3718116 -1.1492030 -3.78283230
 [5,] -1.4330830 -1.1910482 -3.93300606
 [6,] -1.1710980 -1.0910910 -3.83914172
```

```
> table(predict(svm.kernlab, iris), iris[,5])
            setosa versicolor virginica
 setosa         50          0         0
 versicolor      0         47         0
 virginica       0          3        50
```

- predict() 함수를 통해 새로운 자료에 대한 분류(예측)을 수행 할 수 있다. 여기서는 모형 구축에 사용된 자료를 재사용하여 분류를 수행하였다. 그 결과 setosa와 viginica는 50개 모두, versicolor는 50개 중 47개가 제대로 분류되었다.

- 다음의 [예제 3]은 R 패키지 {e1071}의 svm() 함수를 이용하여 서포트벡터회귀(SVR)를 수행한다.

예제 3 svm() 함수를 통해 SVR을 수행하고, 모수 조절(tuning)을 통해 성능을 제고한다.

```
> # 분석용 자료 생성
> x <-c(1:20)
> y <- c(3,4,8,4,6,9,8,12,15,26,35,40,45,54,49,59,60,62,63,68)
> data<-data.frame(x, y)
```

- SVR과의 비교를 위해 lm() 함수를 통해 단순회귀를 적합한다. 그 결과 $RMSE \approx 5.70$을 얻었다.

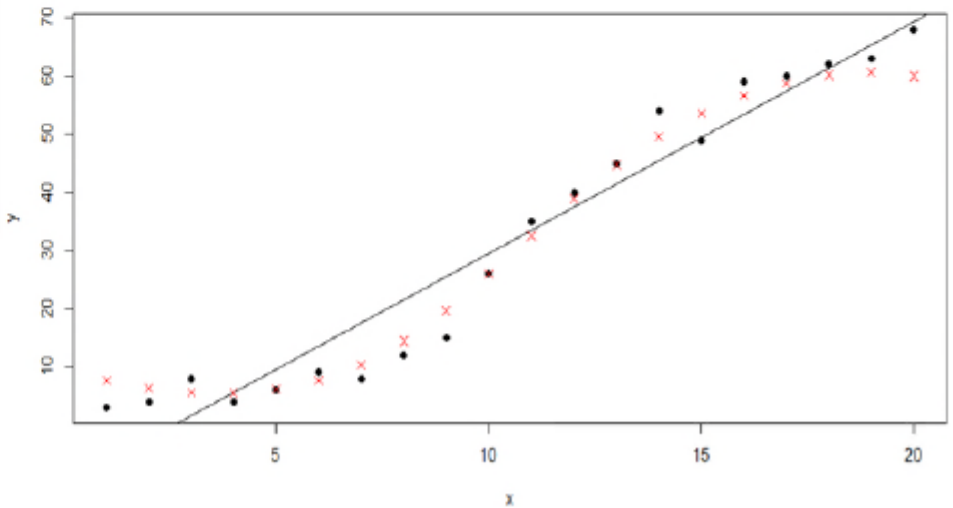
```
> plot(data, pch=16)
> model <- lm(y ~ x, data)
> abline(model)
```

```
> lm.error <- model$residuals # same as data$Y - predictedY
> (lmRMSE <- sqrt(mean(lm.error^2)))
 [1] 5.703778
```

- svm() 함수를 통해 SVR을 수행하고, 그 결과를 단순회귀 결과와 함께 그린다. SVR의 $RMSE \approx 3.16$으로 단순회귀에 비해 감소되었음을 알 수 있다.

```
> model <- svm(y ~ x , data)
> pred.y <- predict(model, data)
> points(data$x, pred.y, col = "red", pch=4)    # pch=4는 'x'임
```

```
> error <- data$y - pred.y
> (svmRMSE <- sqrt(mean(error^2)))
 [1] 3.157061
```



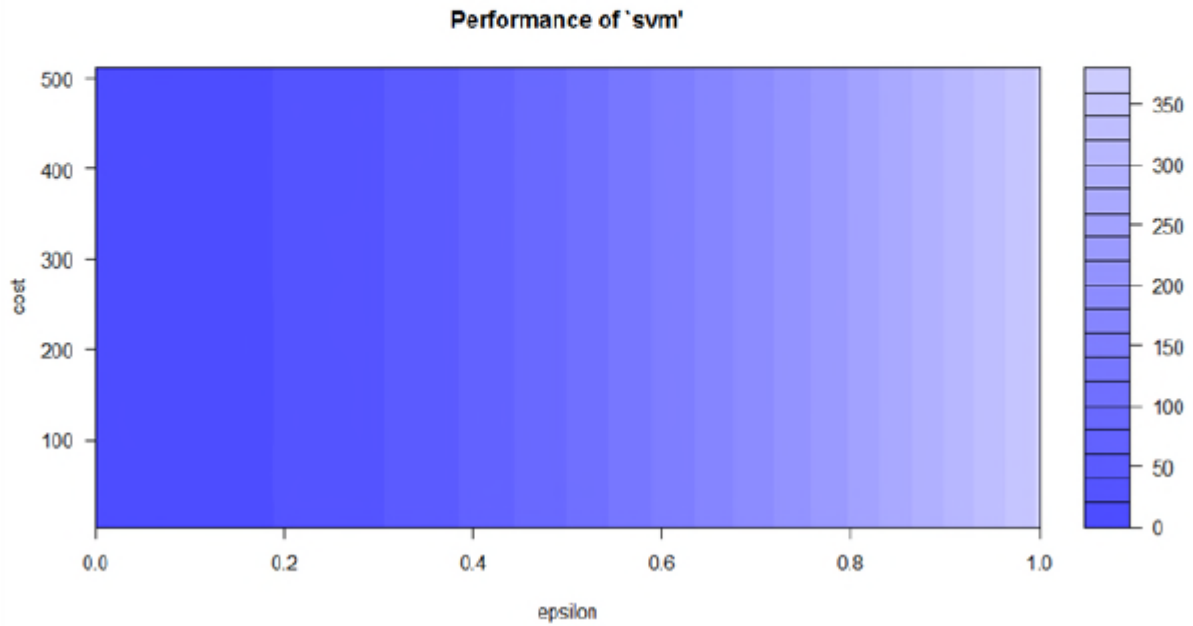
- SVR의 성능을 높이기 위해 모델의 모수들을 최적화 할 필요가 있다. 적용된 svm() 함수는 디폴트로 ε-회귀(type=eps-regression 옵션)가 적용되었으며, ε 값은 디폴트로 0.1이 사용되었다. 과적합(overfitting)을 피하기 위해 cost 모수(cost= 옵션)를 변경할 수 있다.
- 이러한 모수의 선택과정을 초모수 최적화(hyperparameter optimization) 또는 모형 선택(model selection)이라 부른다.
- 모수 ε과 cost의 다양한 값에 대해, 격자탐색을 수행하여, 최적의 모수를 찾는다.
- 이 예제에서는 총 88개의 모형에 대해 훈련이 수행되었다. 그 결과 최적의 모수로 ε=0, cost=256이 선택되었다.

```
> tuneResult <- tune(svm, y ~ x, data = data,
                     ranges = list(epsilon = seq(0,1,0.1), cost = 2^(2:9)))
> # 11×8 = 88 개 모수 조합에서 조율
> print(tuneResult)
 Parameter tuning of ‘svm’:
 - sampling method: 10-fold cross validation
 - best parameters:
 epsilon cost
       0  256
 - best performance: 8.202877
```

- tune() 함수는 각 모형에 대한 MSE 값을 제공한다(제곱근을 취해 RMSE값을 구할 수 있다).

- plot() 함수를 통해 RMSE 값을 시각화 하면 다음과 같다. 이 그림에서 색이 짙을수록 RMSE가 0에 가까우므로 더 나은 모형임을 의미한다.

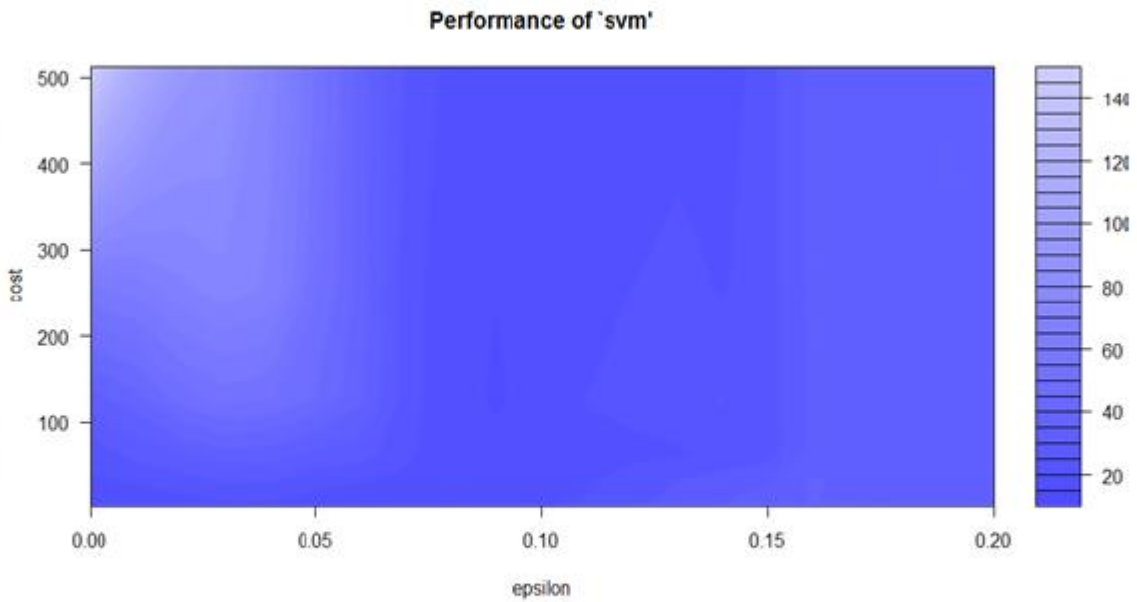
```
> plot(tuneResult)
```



- 보다 정교한 모수 tuning을 위해 보다 좁은 영역에서 격자탐색을 실시한다. 위 그림에서 cost 모수의 영향은 크지 않으므로 그 값을 그대로 유지하였다. 총 168개의 모형에 대해 훈련하였다.

```
> tuneResult <- tune(svm, y ~ x, data=data,
ranges = list(epsilon=seq(0,0.2,0.01), cost=2^(2:9)))    # 168개 모수 조합
> print(tuneResult)
 Parameter tuning of ‘svm’:
 - sampling method: 10-fold cross validation
 - best parameters:
 epsilon cost
    0.09  128
 - best performance: 14.27824
```

```
> plot(tuneResult)
```



- 위의 그림에서 진한 영역을 자세히 볼 수 있으며, 그 결과 ε=0.09, cost=128인 모형이 최소의 오차를 가지는 것을 알 수 있다.

- 다행히도 최적의 모형을 분석자가 찾아낼 필요는 없다. R에서는 다음과 같이 매우 쉬운 방법으로 최적의 모형이 제공되며, 이를 통해 예측을 수행할 수 있다.

```
> tunedModel <- tuneResult$best.model
> tpred.y <- predict(tunedModel, data)
> error <- data$y - tpred.y
> tsvmRMSE <- sqrt(mean(error^2))
> tsvmRMSE
 [1] 2.072399
```

- 위 결과에서 $RMSE \approx 2.07$로 모형의 성능이 크게 개선되었음을 알 수 있다.

- 모수에 대한 조절(tuning) 전과 후의 SVR을 적합한 결과를 그림으로 그려보면 다음과 같다. 그림에서 푸른색은 모수 조절된 SVR 적합 결과이다.

```
> plot(data, pch=16)
> points(data$x, pred.y, col = "red", pch=4, type="b")
> points(data$x, tpred.y, col = "blue", pch=4, type="b")
```