2019 학년도 1학기
DATA MINING
HW8

과목명	데이터마이닝
담당교수명	송종우 교수님
제출일	2019.05.15
학번	182STG27
이름	임지연

Ⅰ. Description

9장에서는 Support Vector Machine에 대해서 공부할 것이다. Maximal Margin classifier, support vector classifier, non-linear한 경우의 support vector machine, class가 3개일 때의 support vector machine, 다른 방법들과의 비교를 해볼 것이다. 각각의 모형에서 사용되는 방법의 원리에 대해서 알아본 후 Lab에 설명되어 있는 코드를 실행해 보며 함수를 공부한 후, 결과를 이해한 후 연습문제를 풀어볼 것이다.

Ⅱ. Implementation

Question 1.

Lab : Support Vector Machines 의 코드를 실행해보고 감상문을 써라

Support vector classifier 방법을 사용하기 위해 e1071 라이브러리에서 svm() 함수를 이용하여 kernel = "linear"로 지정해줬다. 이 함수의 cost 옵션에 대하여 값이 작으면 margin 값이 넓다는 것을 의미하며 즉 support vector가 마진 위에 있을 수 있음을 의미한다.

Support vector machine 방법을 사용하기 위하여 svm() 함수를 이용하며 kernel = "polynomial"를 사용할 경우, 차수(d)를 지정하주며, 또는 kernel = "radial"을 사용할 경우 γ를 지정하여 적절한 커널이 무엇인가를 비교할 수 있다. 만약 여기서 tune() 함수를 사용하면 최적의 cost, γ 값을 찾아주기 때문에 편리하다.

ROC 곡선을 그리기 위하여 ROCR 라이브러리에 내장되어있는 rocplot() 함수를 사용할 수 있다. 여러가지 모형을 비교할 때 ROC커브를 그려본 후 그 아래 면적인 AUC값을 비교할 수 있다.

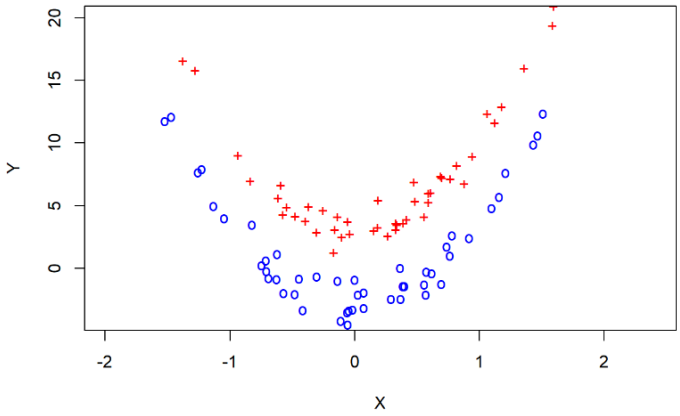
Khan자료에 대해 분석을 하기 전 간단한 차원을 조사하며 반응변수 값에 대한 관측치 수를 봤을 때, 관측치 수는 4개이며 각각 관측치 수는 6개 이하의 값을 갖기 때문에 linear classifier 로 충분함을 알 수 있었다. 따라서 linear kernel을 이용할 수 있음을 미리 알 수 있다는 점에서 모형을 적합할 때, 데이터의 특성에 대한 1차원적인 탐색이 필요하다는 것을 알 수 있었다.

Question 2.

9.7 Excercises - Example 4, 5, 8 풀어라

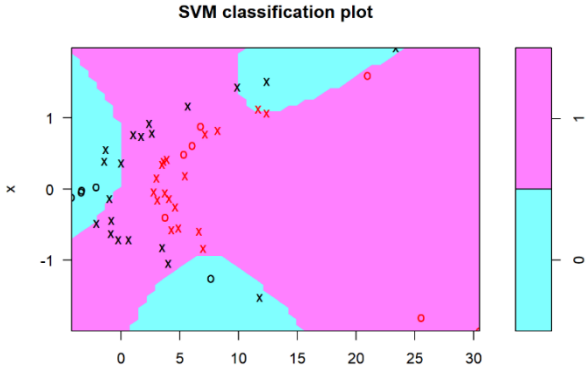
[Example 4]

Generate a simulated two-class data set with 100 observations and two features in which there is a visible but non-linear separation between the two classes. Show that in this setting, a support vector machine with a polynomial kernel (with degree greater than 1) or a radial kernel will outperform a support vector classifier on the training data. Which technique performs best on the test data? Make plots and report training and test error rates in order to back up your assertions.

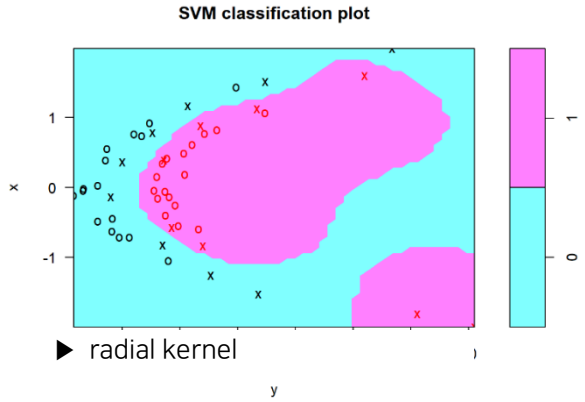


▶ $y = 7x^2$의 식을 가진 비선형 형태를 가진 데이터를 생성 후, 그래프를 그려본 결과이다.

▶ linear kernel

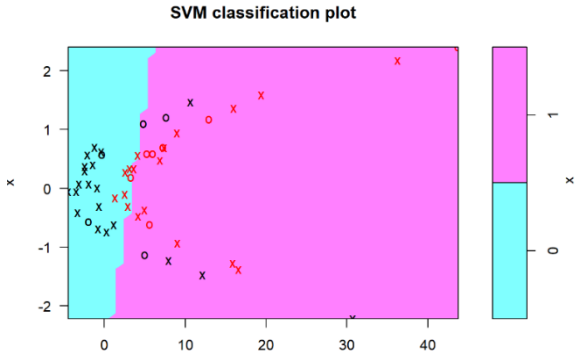


▶ polynomial kernel

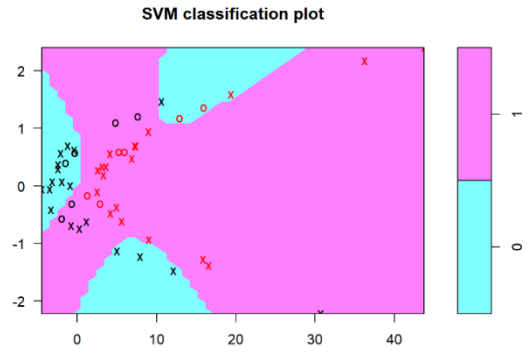


▶ radial kernel

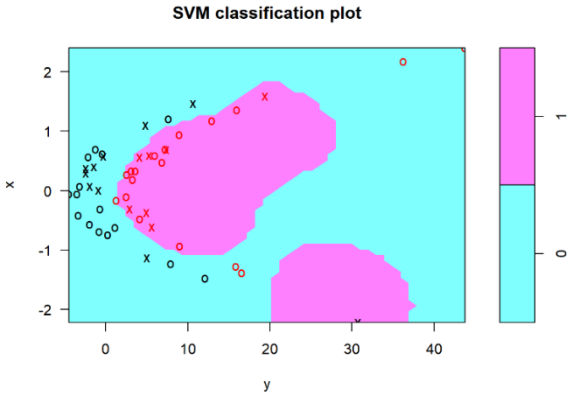
▶ training data 에 linear, polynomial, radial kernel을 각각 적합해 본 결과, 오분류 개수가 각각 10, 15, 0개로 나타났다. 즉, radial kernel이 가장 좋은 성능을 나타냈다.



▶ linear kernel



▶ polynomial kernel



▶ radial kernel

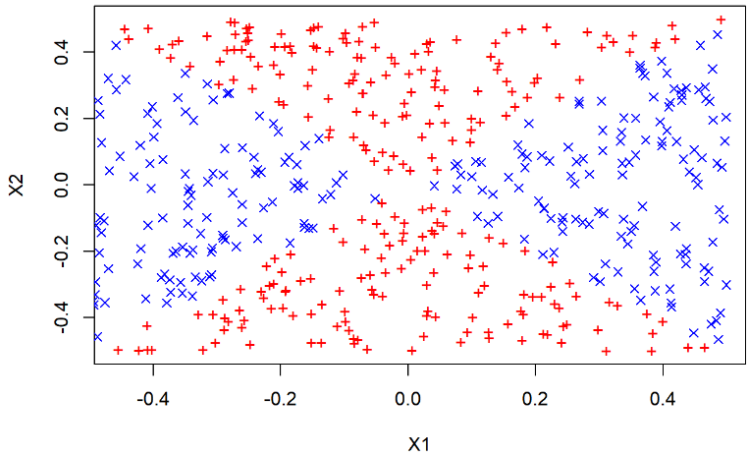
▶ test data 에 linear, polynomial, radial kernel을 각각 적합해 본 결과, 오분류 개수가 각각 11, 11, 5개로 나타났다. 즉, test set과 마찬가지로 radial kernel이 가장 좋은 성능을 나타냈다.

[Example 5]

We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

(a) Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them. For instance, you can do this as follows:

(b) Plot the observations, colored according to their class labels. Your plot should display X1 on the x-axis, and X2 on the yaxis.



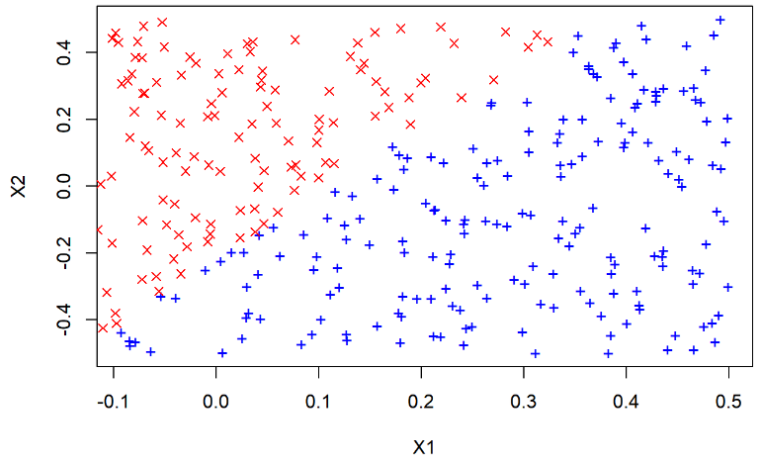
▶ 생성한 데이터는 비선형성을 띄고 있음을 그래프로부터 알 수 있다.

(c) Fit a logistic regression model to the data, using X1 and X2 as predictors.

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-0.08	0.09	-0.84	0.4
x1	0.68	0.32	2.13	0.03
x2	-0.34	0.31	-1.1	0.27

▶ p-value 값을 봤을 때 x1 변수가 유의하다.

(d) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be linear.



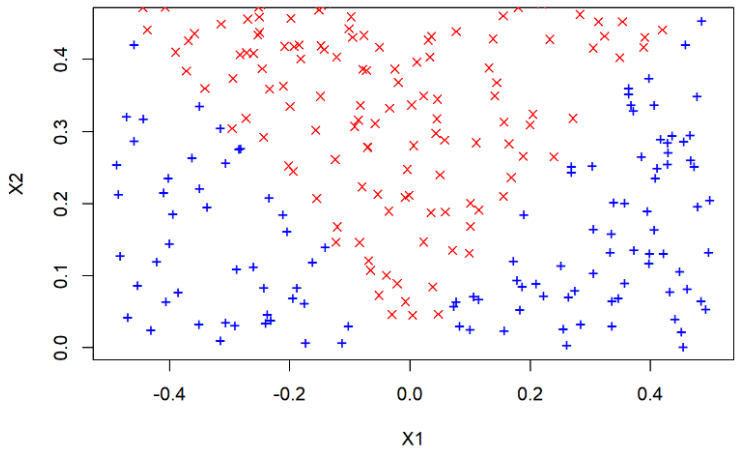
▶ linear kernel 을 이용해 데이터를 적합해 본 결과, 데이터를 적절히 반영하지 못하는 것 같아 보인다.

(e) Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors (e.g. X2 1 , X1×X2, log(X2), and so forth).

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	-240.32	685826.1	0	1
poly(x1, 2)1	2642.37	1803681	0	1
poly(x1, 2)2	14299.48	2388114	0.01	1
poly(x2, 2)1	5719.64	10716860	0	1
poly(x2, 2)2	-17713.1	6340238	0	1
I(x1 * x2)	-215.03	2654498	0	1
log(x2)	8.43	60102.16	0	1

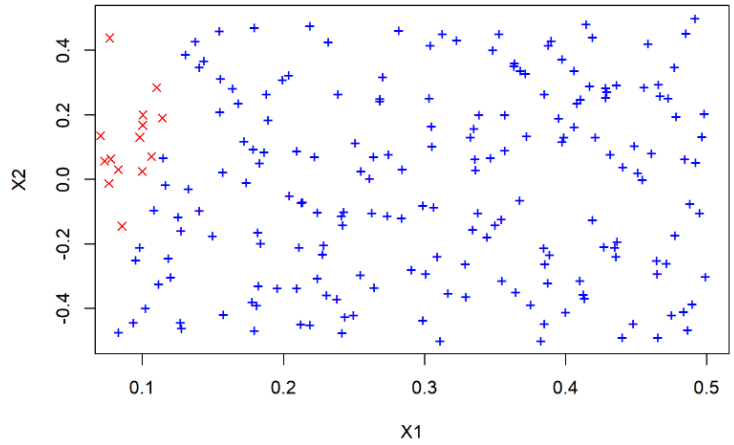
▶ p-value 값을 보면 유의한 변수가 없다.

(f) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an example in which the predicted class labels are obviously non-linear.



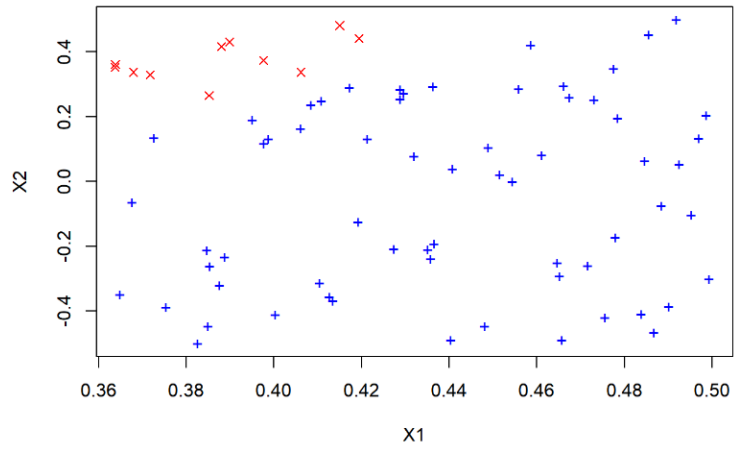
▶ linear kernel 보다는 데이터를 더 적절히 반영하는 듯 보인다.

(g) Fit a support vector classifier to the data with X1 and X2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

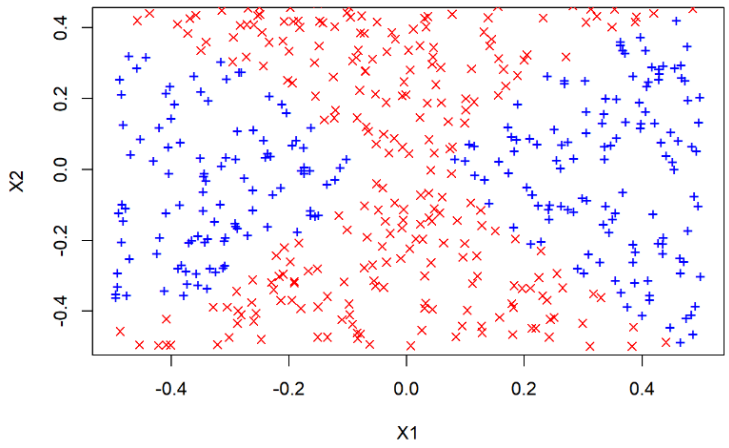


▶ support vector classifie를 적용해본 결과, 데이터를 적절히 반영하지 못하는 것 같아 보인다.

(h) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.



▶ SVM을 Polynomial kernel을 이용해 적용해본 결과, 데이터를 적절히 반영하는 것 같지 않아 보인다.



▶ SVM을 radial kernel을 적용해본 결과, 실제 데이터에서 boundary와 비슷한 형태를 가진다는 것을 알 수 있다.

(i) Comment on your results.

▶ 비선형적인 형태를 가지고 있을 때, 비선형kernel을 적합한 결과가 훨씬 성능이 좋다는 것을 알 수 있었다. 로지스틱, linear kernel을 사용했을 때는 적절한 boundary를 찾지 못하며, 적절한 교호효과 또는 높은 차수를 이용하여 좋은 성능을 낼 수 있지만 비용과 시간의 측면에서 비효율적일 수 있다. 하지만 radial kernel 을 이용하면 적절한 hyperparameter의 조정만으로도 좋은 성능을 나타낸다는 것을 알 수 있었다.

[Example 8]

This problem involves the OJ data set which is part of the ISLR package.

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations. (b) Fit a support vector classifier to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.

```
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear",    cost = 0.01)
## Parameters:
##       SVM-Type:    C-classification
##   SVM-Kernel:    linear
##             cost:    0.01
##           gamma:    0.05555556
## Number of Support Vectors:    432
##   ( 215 217 )
## Number of Classes:    2
## Levels:
##   CH MM
```

▶결과적으로 432개의 support vector가 생성되며 각각 CH가 215개, MM이 217개로 나타난다.

(c) What are the training and test error rates?

▶ training error rate = 0.166, test error rate = 0.181

(d) Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

▶ best cost = 0.0178

(e) Compute the training and test error rates using this new value for cost.

▶ training error rate = 0.16, test error rate = 0.185

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial")
## Parameters:
##      SVM-Type:   C-classification
##   SVM-Kernel:   radial
##           cost:   1
##          gamma:   0.05555556
## Number of Support Vectors:   379
##   ( 188 191 )
## Number of Classes:   2
## Levels:
##   CH MM
```

▶결과적으로 379 개의 support vector가 생성되며 각각 CH 199개, MM 191개로 나타난다.
또한, training error rate = 0.145, test error rate = 0.17로 나타난다.

▶ tune() 함수를 사용해 봤을 때, best cost = 0.56로 나타나며 이 cost 값을 적용해 봤을 때 Training error rate = 0.145, test error rate = 0.167로 나타난다.

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree=2.

```
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "poly", degree = 2)
## Parameters:
##      SVM-Type:   C-classification
##   SVM-Kernel:   polynomial
##           cost:   1
##         degree:   2
##          gamma:   0.05555556
##         coef.0:   0
## Number of Support Vectors:   454
##   ( 224 230 )
## Number of Classes:   2
## Levels:   CH MM
```

- ▶ 454개의 support vector가 생성되며, 각각 CH가 224, MM이 230개로 나타난다. 이때 Training error rate = 0.1725, test error rate = 0.189로 나타난다.
- ▶ tune() 함수를 적용해 봤을 때, best cost = 5.623값으로 나타나며, 이 cost 값을 적용해 봤을 때 Training error rate = 0.147, test error rate = 0.185로 나타난다.

(h) Overall, which approach seems to give the best results on this data?

- ▶ linear, polynomial, radial kernel을 적용해 봤을 때, training error rate, test error rate 값이 polynomial 일 경우가 가장 성능이 좋았다.

Ⅲ. Discussion

p차원에서 Hyperplane이란 차원이 p-1인 affine공간을 의미한다. 데이터를 몇 개의 공간으로 나누기 위해서 Hyperplane의 개념이 도입되는 것이며, 반응변수의 클래스 라벨에 따라서 분리하게 된다. 데이터가 Hyperplane과의 거리가 멀수록 확실하게 분리된다는 의미로 해석 가능하다. 이 Hyperplane로 인해 나뉜 공간들 중 Hyperplane과의 거리가 가장 가까운 점과의 거리를 margin이라고 하는데 이 margin이 가장 크게 하는 것이 최적의 margin classifier 가 된다.

Support vector classifier는 경계가 선형인 경우 사용할 수 있는 방법이다. Support vector machine은 경계가 비선형인 경우 사용하기 좋은 방법인데, kernel을 사용하여 변수공간을 확장하는 방법이다. 비선형인 경우 대체로 radial kernel 의 성능이 가장 좋은 것을 알 수 있었다. 모형적합 후 Plot을 그려 비교해 보며 다른 방법들과의 성능을 비교할 수 있었다.

Ⅳ. Appendix – R code

```
## 4번 문제
```{r }
library(e1071)
### data setting ###
# train set
set.seed(1)
x = rnorm(100)
y = 7*x^2 + rnorm(100)
train = sample(100, 50)
y[train] = y[train] + 3
y[-train] = y[-train] - 3
plot(x[train], y[train], pch="+", lwd=4, col="red", ylim=c(-4, 20), xlab="X", ylab="Y")
points(x[-train], y[-train], pch="o", lwd=4, col="blue")
# split variable z
set.seed(315)
z = rep(0, 100)
z[train] = 1
final.train = c(sample(train, 25), sample(setdiff(1:100, train), 25))
# train, test
data.train = data.frame(x=x[final.train], y=y[final.train], z=as.factor(z[final.train]))
data.test = data.frame(x=x[-final.train], y=y[-final.train], z=as.factor(z[-final.train]))
```
```

```
### train set - 3 kernel fit ###
# linear kernel
svm.linear = svm(z~., data=data.train, kernel="linear", cost=10)
plot(svm.linear, data.train)
linear.table = table(z[final.train], predict(svm.linear, data.train))
linear.table
linear.table[1,2] + linear.table[2,1] # number of error
# polynomial kernel
set.seed(1)
svm.poly = svm(z~., data=data.train, kernel="polynomial", cost=10)
plot(svm.poly, data.train)
poly.table = table(z[final.train], predict(svm.poly, data.train))
poly.table
poly.table[1,2] + poly.table[2,1] # number of error
# radial kernel
set.seed(1)
svm.radial = svm(z~., data=data.train, kernel="radial", gamma=1, cost=10)
plot(svm.radial, data.train)
radial.table = table(z[final.train], predict(svm.radial, data.train))
radial.table
radial.table[1,2] + radial.table[2,1] # number of error # perfect !!!!
### test set - 3 kernel fit ###
plot(svm.linear, data.test)
test1 = table(z[-final.train], predict(svm.linear, data.test))
test1
test1[1,2] + test1[2,1]
plot(svm.poly, data.test)
test2 = table(z[-final.train], predict(svm.poly, data.test))
test2
test2[1,2] + test2[2,1]
plot(svm.radial, data.test)
test3 = table(z[-final.train], predict(svm.radial, data.test))
test3
test3[1,2] + test3[2,1]
```

5번 문제
```{r }
# a
set.seed(400)
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = 1 * (x1^2 - x2^2 > 0)
# b
plot(x1[y == 0], x2[y == 0], col = "red", xlab = "X1", ylab = "X2", pch = "+")
points(x1[y == 1], x2[y == 1], col = "blue", pch = 4)
# non-linear !!!!
# c
lm.fit = glm(y ~ x1 + x2, family = binomial)
summary(lm.fit)
# x1, x2 : insignificant!!!
# d
data = data.frame(x1 = x1, x2 = x2, y = y)
lm.prob = predict(lm.fit, data, type = "response")
lm.pred = ifelse(lm.prob > 0.5, 1, 0)
data.pos = data[lm.pred == 1, ]
data.neg = data[lm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
# linear boundary!!!
# e
lm.fit = glm(y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2) + log(x2), data = data, family = binomial)
summary(lm.fit)
# f
lm.prob = predict(lm.fit, data, type = "response")
lm.pred = ifelse(lm.prob > 0.5, 1, 0)
data.pos = data[lm.pred == 1, ]
data.neg = data[lm.pred == 0, ]
```

```
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
# non-linear boundary !!! d보다는 잘 예측한다
# g
library(e1071)
svm.fit = svm(as.factor(y) ~ x1 + x2, data, kernel = "linear", cost = 0.1)
svm.pred = predict(svm.fit, data)
data.pos = data[svm.pred == 1, ]
data.neg = data[svm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
# h
# polynomial
svm.fit = svm(as.factor(y) ~ x1 + x2, data, kernel="polynomial", cost=5)
svm.pred = predict(svm.fit, data)
data.pos = data[svm.pred == 1, ]
data.neg = data[svm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
#radial
svm.fit = svm(as.factor(y) ~ x1 + x2, data, kernel = "radial", gamma = 1)
svm.pred = predict(svm.fit, data)
data.pos = data[svm.pred == 1, ]
data.neg = data[svm.pred == 0, ]
plot(data.pos$x1, data.pos$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(data.neg$x1, data.neg$x2, col = "red", pch = 4)
# goooooooood !! 실제 값에서의 경계와 유사함

#i

```
8번 문제
```{r }

# a
library(ISLR)
set.seed(1)
train = sample(dim(OJ)[1], 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
# b
library(e1071)
svm.linear = svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = 0.01)
summary(svm.linear)
# 432개 support vector 생성, CH:215개  MM:217개
# c
train.pred = predict(svm.linear, OJ.train)
test.pred = predict(svm.linear, OJ.test)
t1 =table(OJ.train$Purchase, train.pred)
t2 =table(OJ.test$Purchase, test.pred)
(t1[1,2] + t1[2,1])/sum(t1)   # training error rate
(t2[1,2] + t2[2,1])/sum(t2)   # test error rate
# d
set.seed(1)
tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear",
                ranges = list(cost = 10^seq(-2,1, by = 0.25)))
summary(tune.out)
# optimal cost = 0.01778279

# e
svm.linear = svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = tune.out$best.parameters$cost)
train.pred = predict(svm.linear, OJ.train)
test.pred = predict(svm.linear, OJ.test)
t1 =table(OJ.train$Purchase, train.pred)
t2 =table(OJ.test$Purchase, test.pred)

(t1[1,2] + t1[2,1])/sum(t1)   # training error rate
(t2[1,2] + t2[2,1])/sum(t2)   # test error rate
```

```
# f
# radial kernel 이용
set.seed(1)
svm.radial = svm(Purchase ~ ., data = OJ.train, kernel = "radial")
summary(svm.radial)

train.pred = predict(svm.radial, OJ.train)
f1 = table(OJ.train$Purchase, train.pred)
test.pred = predict(svm.radial, OJ.test)
f2 = table(OJ.test$Purchase, test.pred)

(f1[1,2]+f1[2,1])/sum(f1) # training error rate
(f2[1,2]+f2[2,1])/sum(f2) # test error rate

# tune 함수 사용
set.seed(755)
tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list(cost = 10^seq(-2,
    1, by = 0.25)))
summary(tune.out)

svm.radial = svm(Purchase ~ ., data = OJ.train, kernel = "radial", cost = tune.out$best.parameters$cost)
train.pred = predict(svm.radial, OJ.train)
test.pred = predict(svm.radial, OJ.test)

f3 = table(OJ.train$Purchase, train.pred)
f4 = table(OJ.test$Purchase, test.pred)

(f3[1,2]+f3[2,1])/sum(f3) # training error rate
(f4[1,2]+f4[2,1])/sum(f4) # test error rate


# g
# poly kernel , degree=2이용
set.seed(1)
svm.poly = svm(Purchase ~ ., data = OJ.train, kernel = "poly", degree = 2)
summary(svm.poly)

train.pred = predict(svm.poly, OJ.train)
test.pred = predict(svm.poly, OJ.test)

g1 = table(OJ.train$Purchase, train.pred)
g2 = table(OJ.test$Purchase, test.pred)
(g1[1,2]+g1[2,1])/sum(g1) # training error rate
(g2[1,2]+g2[2,1])/sum(g2) # test error rate

# tune 함수
set.seed(322)
tune.out = tune(svm, Purchase ~ ., data = OJ.train, kernel = "poly", degree = 2,
    ranges = list(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune.out)

svm.poly = svm(Purchase ~ ., data = OJ.train, kernel = "poly", degree = 2, cost = tune.out$best.parameters$cost)
train.pred = predict(svm.poly, OJ.train)
test.pred = predict(svm.poly, OJ.test)

g3 = table(OJ.train$Purchase, train.pred)
g4 = table(OJ.test$Purchase, test.pred)
(g3[1,2]+g3[2,1])/sum(g3) # training error rate
(g4[1,2]+g4[2,1])/sum(g4) # test error rate


# h


```