Data Mining

HW#8: Support Vector Machines

학번	182STG18
이름	이하경
제출일	2019.05.15

Description

Support Vector Machines

두 가지 클래스 분류 문제에서 가장 직접적인 방법은 클래스들을 구분하는 설명변수들의 공간을 가능한 만큼 정확히 분리(separate)하는 방법을 찾는 것이다. 데이터가 p 차원일 때 이 공간을 분리하는 hyperplane 은 p-1 차원으로 형성된다. 모든 가능한 separating hyperplane 중에서 클래스 간에 가장 큰 gap(또는 margin)을 찾는 Maximal Margin Classifier 를 고려할 수 있다.

$$\text{Maximize M (margin) subject to } \sum_{j=1}^{p} \mathrm{B}_j^2 ,$$
$$\mathrm{y_i}\left(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}\right) \geq M \; for \; all \; i = 1, \ldots, N$$

실제 많은 경우 관측치를 정확히 분류하는 hyperplane 을 찾는 것은 불가능하므로, 잘못 분류되는 관측치들의 decision boundary 까지의 거리 ϵ_i의 총합이 상수 C 보다 작도록 tuning parameter C 를 고려할 수 있다. 또한 클래스 별 관측치들의 구분이 선형으로 가능하지 않은 경우 변수의 차수를 높여 (ex. $\mathrm{X}_1^2, \mathrm{X}_1 X_2, \ldots$) 공간을 확장하고 그 곳에서 선형 support vector classifier 를 찾는다. 이 결과는 원래의 설명변수 공간을 non-linear decision boundary 로 구분하는 방법이다.

또한 변수에 radial kernel $\mathrm{K}(\mathrm{x_i}, \mathrm{x_i'}) = \exp\left(-\gamma \sum_{j=1}^{p}\left(x_{ij} - x_{ij}'\right)^2\right)$을 적용해 비선형 boundary 를 적합할 수 있다. 이 경우 γ의 크기에 따라 boundary 가 smooth 한 정도를 조절할 수 있는데, 이 역시 tuning parameter 중 하나이다.

본 과제에서는 SVM 의 Tutorial 및 ISLR 의 Lab 을 통해 R 에서 SVM 을 적합하고 결과를 확인하는 실습을 해보고 총 3 가지 예제에서 시뮬레이션으로 생성한 데이터 또는 실제 데이터에 다양한 SVM 모형을 적용해본다.

Results

Chapter 9 Lab & SVM Tutorial

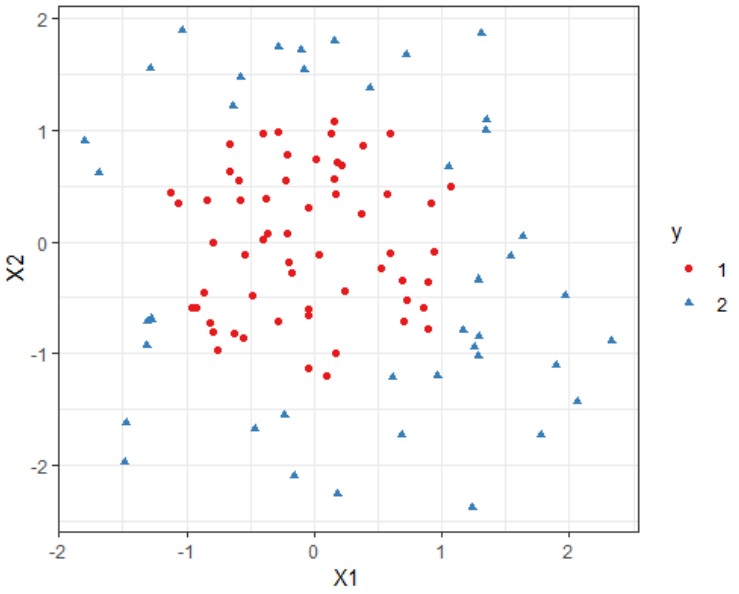
R의 'e1071' 또는 'kernlab' 패키지를 설치하여 SVM 모형을 적용할 수 있다. 먼저 정규분포에서 랜덤하게 두 개의 설명변수를 구성하고 선형으로 클래스들이 구분되도록 시뮬레이션 데이터를 만들었다. 여기에 **svm** (또는 ksvm) 함수를 이용해 linear kernel svm 모형을 적합해보았다. SVM은 boundary로부터 관측치들의 거리를 계산하므로 각 설명변수가 거리에 기여하는 정도가 동일하길 기대하며, 그러므로 변수들의 크기를 정규화하는 것이 좋다. (scale=TRUE). **tune**() 함수를 사용해 설정한 cost의 범위에 따라 cross-validation을 진행하고 CV error 가 가장 작은 cost parameter의 값을 찾아보았다.

비슷하게 랜덤하게 구성한 데이터에서 클래스 별 구분을 비선형으로 만들고, boundary가 linear가 아닌 radial 또는 polynomial 모형을 적합하고 linear 모형과의 퍼포먼스를 비교하였다. 적합한 모형을 시각적으로 확인하기 위해서 plot을 그려보았다.

이를 통해 R에서 SVM을 수행하기 위한 각종 함수와 옵션 설정에 대해 익숙해지게 되었다.

Exercise 9.4

Generate a simulated 2-class data set with 100 observations and 2 features in which there is a visible but non-linear separation between the 2 classes.



Normal(0, 1)에서 각각 100 개의 X_1과 X_2를 랜덤하게 생성하고 $X_1^2 + X_2^2 \leq 1.5$이면 클래스를 1, 그렇지 않으면 클래스를 2 로 분류하였다. 따라서 실제로 클래스가 구분되는 경계선은 원 모양이 된다.

생성된 총 100 개의 관측치 중 각 클래스별 분포는 다음과 같다.

	class 1	class 2
nobs	59	41

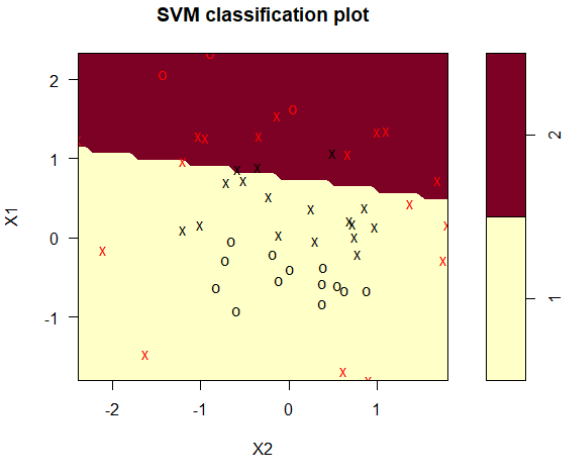
Show that in this setting, a support vector machine with a polynomial kernel (with degree greater than 1) or a radial kernel will outperform a support vector classifier on the training data. Which technique performs best on the test data? Make plots and report training and test error rates in order to back up your assertions.

100 개의 관측치를 각각 50 개, 50 개의 training data 와 test data 로 분할하고 training data 만을 사용해서 각각의 SVM 모형을 적합하였다. 먼저 linear kernel 을 사용해 모형을 적합하고, 같은 training data 에 radial, polynomial kernel 을 사용한 모형과 training data 에 대한 성능을 비교하였다. 세 가지 모형에서 튜닝 파라미터인 cost 는 10 으로 설정하였다.

1) Linear kernel

▷ Summary

SVM-Type	C-classification
SVM-Kernel	linear
cost	10
Number of Support Vectors	34 (17 17)



▷ Confusion Matrix of Training data (Training Error Rate=20%)

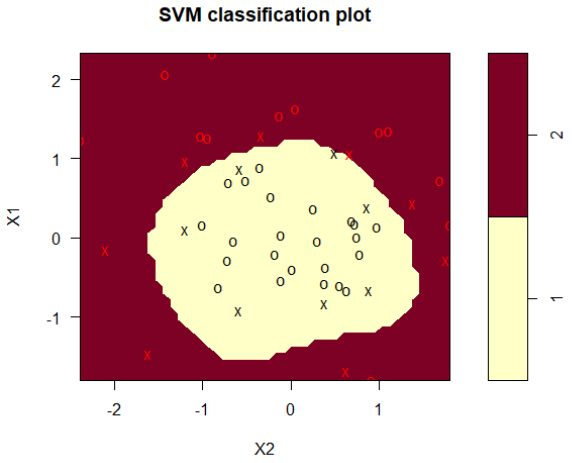
	Fitted	
TRUE	1	2
1	27	3
2	7	13

총 34 개의 관측치가 decision boundary 를 결정하는 support vectors 로 사용되었고 50 개의 training 관측치 중 오분류된 관측치의 개수는 10 개이다.

2) Radial kernel

▷ **Summary**

SVM-Type	C-classification
SVM-Kernel	radial
cost	10
gamma	1
Number of Support Vectors	15 (8 7)



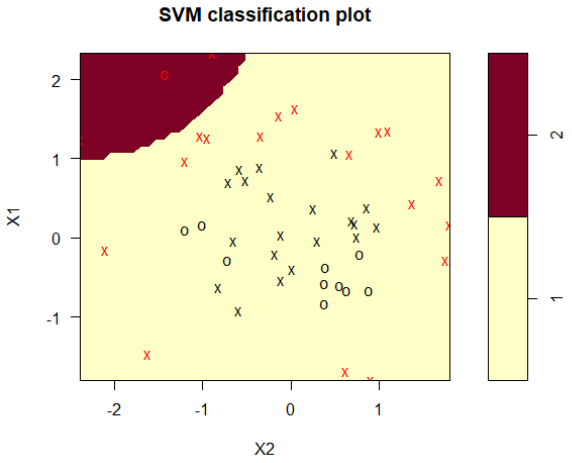
▷ **Confusion Matrix of Training data (Training Error Rate=0%)**

	Fitted	
TRUE	1	2
1	30	0
2	0	20

radial kernel 의 경우 gamma 를 1 로 설정하였다. decision boundary 결정에 사용된 관측치의 개수가 더 적으면서 classification plot 을 보면 실제 경계선의 모양에 가깝게 원 모양을 하고 있는 것을 확인할 수 있다. training 관측치 50 개를 모두 정확히 분류하여 오분류율은 0%로 나타났다.

3) Polynomial kernel

▷ **Summary**



SVM-Type	C-classification
SVM-Kernel	polynomial
cost	10
degree	3
gamma	1
coef.0	0
Number of Support Vectors	39 (19 20)

▷ **Confusion Matrix of Training data (Training Error Rate=0%)**

	Fitted	
TRUE	1	2
1	30	0
2	17	3

degree 가 3 인 polynomial kernel 의 경우, decision boundary 를 결정하는 support vector 의 개수가 39 개로 제일 많다. 실제 클래스가 1 인 관측치들은 모두 1 로 분류하지만 실제 클래스가 2 인 관측치 중 일부만 옳게 분류한 것을 그림을 통해 확인할 수 있다. training error 는 34%로 세 가지 모형 중 가장 크게 나타났다. 시뮬레이션으로 생성한 본 예제의 데이터의 경우 polynomial kernel 을 적합하는 것은 적합하지 않다고 볼 수 있다.

▶ **Test Error Rate in 3 Models**

위의 3 가지 모형으로 50 개의 test data 에 대한 prediction 을 진행한 뒤 실제 클래스와 비교해 오분류율을 구했을 때 결과는 다음과 같다.

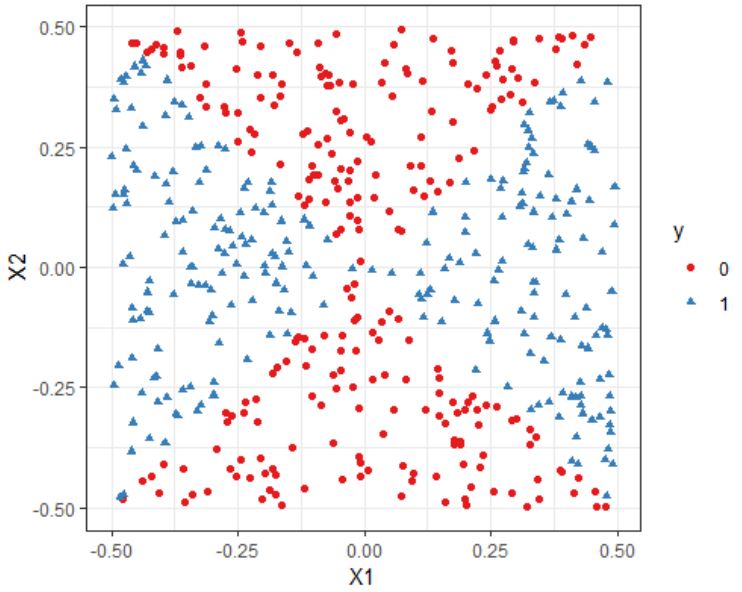
	linear	radial	polynomial
test	**38%**	**10%**	**38%**
train	20%	0%	34%

세 모형 모두 test error 가 train error 보다 크게 계산되었지만 오분류율이 0 으로 성능이 가장 좋았던 radial kernel 모형의 경우가 test error 역시 가장 작아 퍼포먼스가 가장 좋았다.

Exercise 9.5

(a) Generate a data set with n=500, p=2, such that the observations belong to 2 classes with a quadratic decision boundary between them.

(b) Plot the observations, colored according to their class labels.



X_1과 X_2 모두 Uniform(-0.5, 0.5)에서 500개씩 생성하고 $X_1^2 > X_2^2$ 인 경우 클래스를 1, 그렇지 않은 경우 0으로 분류하였다. quadratic decision boundary를 확인할 수 있다.

(c) Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

▷ **Summary**

Call: glm(formula = y ~ X1 + X2, family = binomial)

Coefficients:

	Estimate	Std. Error	z-value	P(>|z|) (p-value)
(Intercept)	0.0316	0.0897	0.353	0.724
X1	-0.1436	0.2998	-0.479	0.632
X2	0.4683	0.3138	1.492	0.136

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 693.02 on 499 degrees of freedom

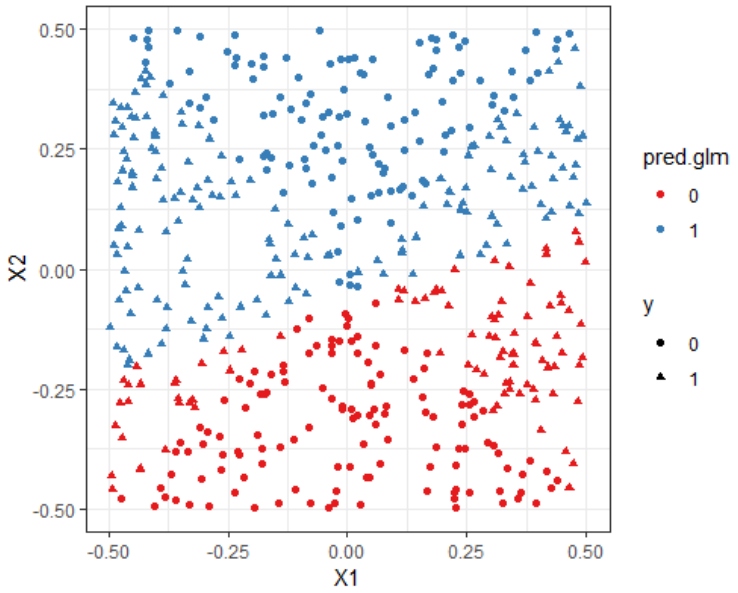
Residual deviance: 690.53 on 497 degrees of freedom

AIC: 696.53

Number of Fisher Scoring iterations: 3

X1과 X2 모두 클래스를 설명하기에 유의하지 않다.

(d) Plot the observations, colored according to the predicted class labels by the model. The decision boundary should be linear.



GLM 모형에서 training data에 대해 예측된 확률이 0.5보다 큰 경우 클래스를 1로, 작은 경우 0으로 분류하였다. 직선 모양을 경계선으로 두 클래스가 분류되어 있음을 확인할 수 있다. 그림에서 점들의 모양은 실제 값에 따라 구분되어 있다.

(e) Fit a regression model to the data using non-linear functions of X_1 and X_2 as predictors.

▷ **Summary**

Call: glm(formula = y ~ poly(X1, 2) + poly(X2, 2), family = binomial)

Coefficients:

	Estimate	Std. Error	z-value	P(>|z|) (p-value)
(Intercept)	356.9	7690.6	0.046	0.963
poly(X1, 2)1	-5609.5	107889.2	-0.052	0.959
poly(X1, 2)2	76421.6	1214465.3	0.065	0.948
poly(X2, 2)1	1627.8	43826.5	0.037	0.970
poly(X2, 2)2	-74978.4	1147713.5	-0.065	0.948

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 693.02 on 499 degrees of freedom

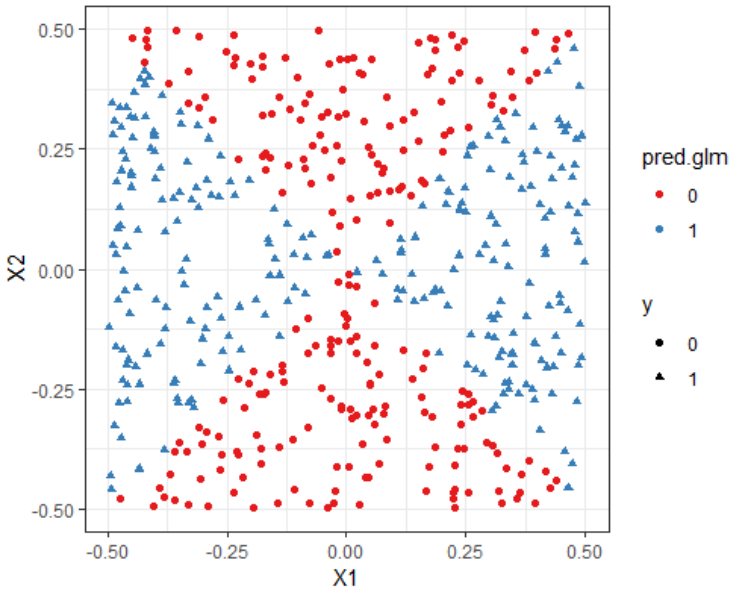
Residual deviance: 1.97e-05 on 497 degrees of freedom

AIC: 10

Number of Fisher Scoring iterations: 25

X_1과 X_2의 2차항을 모두 포함한 비선형 모형을 적합하였으나, 모든 1차항과 2차항의 계수가 유의하지 않았다.

(f) Plot the observations, colored according to the predicted class labels by the non-linear model. The decision boundary should be obviously non-linear.



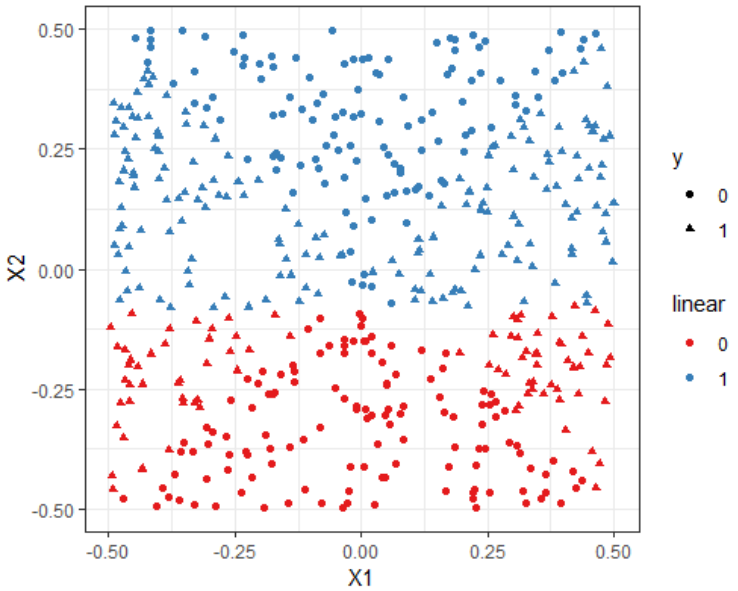
위의 비선형 모형에서 개별 설명변수의 항들은 설명력이 없으나 적합된 클래스로 점들을 구분해서 그림을 그렸을 때 실제 클래스를 정확히 추정하였다. decision boundary가 선형이 아닌 것을 명백하게 확인할 수 있다.

(g) Fit a Support Vector Classifier to the data. Obtain a class prediction, and plot the observations.

▷ **Summary**

SVM-Type	C-classification
SVM-Kernel	linear
cost	1
gamma	0.5
Number of Support Vectors	478 (239 239)

linear kernel을 사용한 Classifier 모형으로 training data에 대해 예측된 분류 그림에서 linear decision boundary를 확인할 수 있다. 많은 관측치들이 잘못 분류되었다.

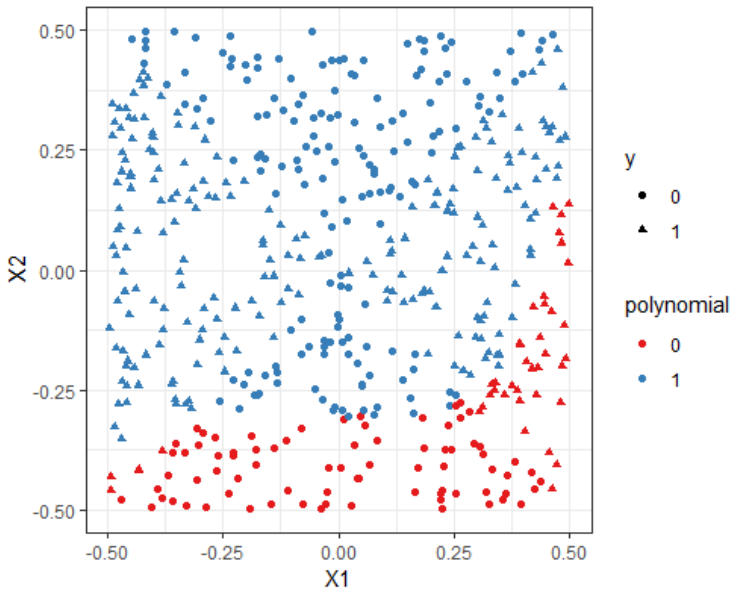


(h) Fit a SVM using a non-linear kernel to the data. Obtain a class prediction, and plot the observations.

non-linear kernel으로서 polynomial kernel과 radial kernel을 사용한 두 가지 SVM 모형을 적합하였다.

▷ **Summary of Polynomial kernel SVM**

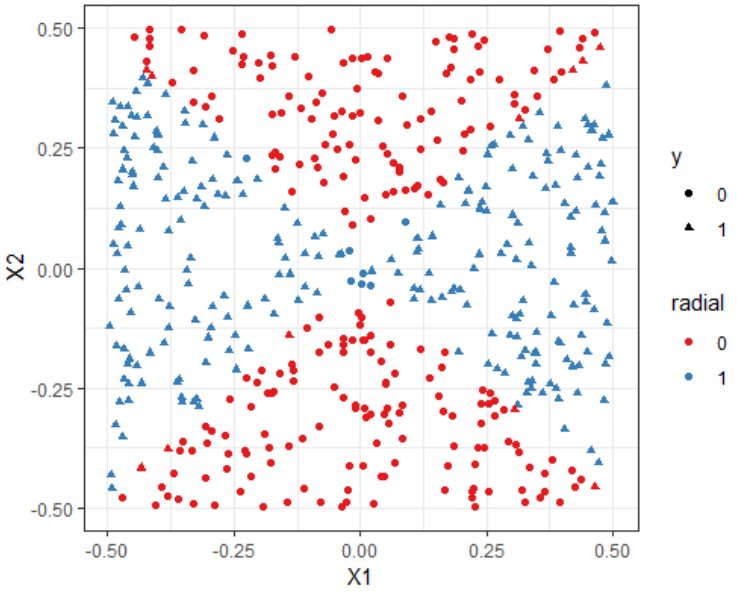
SVM-Type	C-classification
SVM-Kernel	polynomial
cost	1
degree	3
gamma	0.5
coef.0	0
Number of Support Vectors	485 (243 242)



3차의 polynomial kernel을 사용한 SVM 모형에서 비선형 decision boundary가 적용되었으나 경계가 적절하지 않음을 확인할 수 있다.

▷ **Summary of Radial kernel SVM**

SVM-Type	C-classification
SVM-Kernel	radial
cost	1
gamma	0.5
Number of Support Vectors	183 (93 90)



radial non-linear kernel을 사용하였을 때 cost를 1, gamma를 0.5로 정한 결과 완벽히는 아니지만 대부분의 클래스들이 올바르게 분류된 것을 확인할 수 있었다.

(i) Comment on your results.

이 예제에서 생성한 랜덤 데이터의 경우 polynomial logit 모형을 사용하거나 SVM에서 radial kernel을 사용을 할 때 training data에 대한 분류의 정확도가 높다. SVM 모형에서 polynomial kernel을 사용한 경우보다 polynomial logit 모형으로 비선형 decision boundary를 더 정확히 추정할 수 있었다. 위에서 적합한 모든 모형에 대해 training data에 대한 오분류율은 다음과 같다. non-linear logit 모형, 즉 X1과 X2의 2차항을 모두 도입한 polynomial 모형에서 training 오분류율이 0으로 가장 좋다.

linear logit	non-linear logit	linear SVM	polynomial SVM	radial SVM
41.8%	0.00%	40.8%	42.0%	3.8%

Exercise 9.8 OJ data

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

총 1070 개의 관측치 중 800 개의 랜덤한 관측치로 이루어진 training set 과 나머지 test set 이 생성되었다.

(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.

▷ **Summary Statistics**

SVM-Type	C-classification
SVM-Kernel	linear
cost	0.01
gamma	0.0556
Number of Support Vectors	435 (218 217)

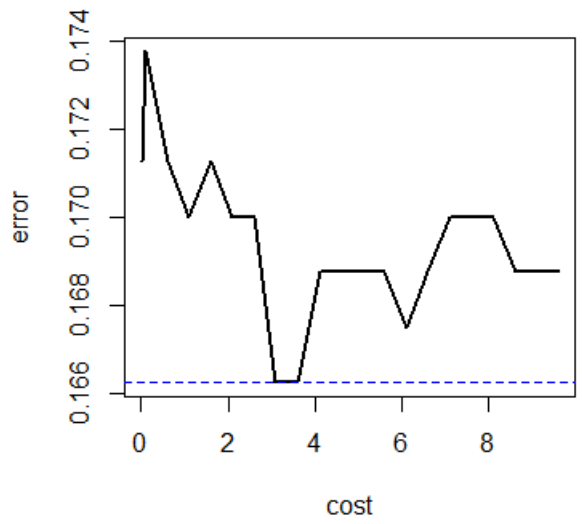
training data 에 대해 linear kernel 을 사용한 support vector classifier 를 적합하였다. gamma 에 값을 지정하지 않은 경우 default 로 지정되는 값은 1/(data dimension)으로, OJ 데이터에서 변수의 개수는 반응변수를 포함하여 총 18 개이므로 1/18=0.0556 으로 설정되었다. linear decision boundary 를 결정하는 데 사용된 training 관측치의 개수는 총 435 개로 클래스 CH 인 관측치 218 개와 MM 인 관측치 217 개이다.

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

Error Rate=16.38%			**Error Rate=17.04%**		
Train	Fitted		**Test**	Fitted	
TRUE	1	2	TRUE	1	2
1	437	50	1	149	17
2	81	232	2	29	75

(d) Tune the model to select an optimal cost. Consider values in the range 0.01 to 10.

10-fold cross validation 을 이용해 0.01 부터 10 까지의 범위에 대해서 cv-error 를 가장 작게 하는 cost 의 값을 찾아보았다.



결과는 다음과 같이 cost=3.1 에서 test error 가 약 16.63%로 최적의 cost 값으로 선택되었다.

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

Error Rate=16.00%			Error Rate=17.04%		
Train	Fitted		**Test**	Fitted	
TRUE	1	2	TRUE	1	2
1	432	55	1	148	18
2	73	240	2	28	76

cost=3.1 으로 training 과 test set 에 대한 오분류율을 계산하였을 때 training error 는 좀 더 감소하였지만 test error 의 크기는 동일하였다.

(f) Repeat (b)-(e) using a SVM with a radial kernel. Use the default value for gamma.

▷ **Summary Statistics**

SVM-Type	C-classification
SVM-Kernel	radial
cost	0.01
gamma	0.0556
Number of Support Vectors	627 (314 313)

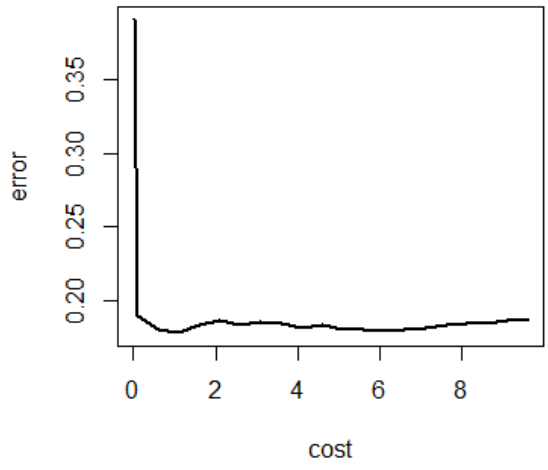
radial kernel 을 사용한 경우 Support Vector 의 개수가 linear kernel 모형보다 많았다.

▷ **training and test error rates (cost=0.01)**

Error Rate=39.13%			Error Rate=38.52%		
Train	Fitted		**Test**	Fitted	
TRUE	1	2	TRUE	1	2
1	487	0	1	166	0
2	313	0	2	104	0

cost=0.01 이고 gamma 의 값은 default 로 적용하였을 때 confusion matrix 를 보면 training set 과 test set 모두 모든 관측치에 대해 클래스를 1 로 분류한 것을 확인할 수 있었다.

▷ **cost Tuning**



선택된 cost parameter 의 값은 1.1 으로 이때 test error 는 17.75 로 나타난다. 이 값을 이용해 training 과 test set 에 대한 error 를 다시 계산하였을 때 오분류율은 각각 **15.25%, 16.30%**로 계산되었다.

(g) Repeat (b)-(e) using a SVM with a polynomial kernel. Set degree=2.

▷ **Summary Statistics**

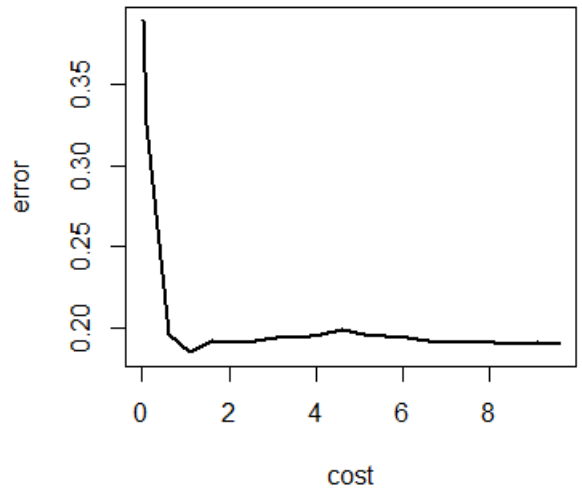
SVM-Type	C-classification
SVM-Kernel	radial
cost	0.01
degree	2
gamma	0.0556
coef.0	0
Number of Support Vectors	633 (320 313)

▷ **training and test error rates (cost=0.01)**

Error Rate=36.88%			**Error Rate=37.04%**		
Train	Fitted		**Test**	Fitted	
TRUE	1	2	TRUE	1	2
1	484	3	1	166	0
2	292	21	2	100	4

2 차항까지 고려한 polynomial kernel SVM 모형의 경우 cost=0.01 이고 gamma 의 값은 default 로 적용하였을 때 confusion matrix 를 보면 radial kernel 모형에 비해 클래스 2 의 관측치를 조금 더 올바르게 분류한 것을 확인할 수 있다.

▷ **cost Tuning**



polynomial kernel 에서 10-folds cross validation 의 결과는 radial kernel 과 크게 다르지 않았다. 0.01 부터 10 까지의 cost 중 선택된 값은 1.1 으로 radial kernel 과 동일하다. 이 값을 이용해 training 과 test set 에 대한 error 를 다시 계산하였을 때 오분류율은 각각 **17.38%, 19.26%**으로 계산되었다.

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

단위: %	**linear**	**radial**	**poly**	**best.linear**	**best.radial**	**best.poly**
train	16.38	39.13	36.88	16.00	15.25	17.38
test	17.04	38.52	37.04	17.04	**16.30**	19.26
CV	**16.63**	17.75	18.5	-	-	-

training set 과 test set 에 대해 각 kernel 에서 cv-error 기준 최적 cost 로 적합된 모형을 이용해 error rate 를 계산한 결과이다. 10-folds CV 결과 linear kernel 의 best performance 가 가장 좋으며 parameter tuning 을 진행한 결과 OJ data 에 대해서는 radial kernel 을 사용했을 때 테스트 데이터에 오분류율이 가장 낮음을 확인할 수 있다.

Discussion

다양한 예제를 통해 클래스 분류 문제에서 SVM 모형을 사용해보았다. cost 또는 gamma 파라미터의 tuning 을 통해 단순히 임의 설정하는 것보다 예측 퍼포먼스가 뛰어난 모형을 찾을 수 있으므로 실제 분류 문제에서 tuning 을 꼭 하는 것이 좋다는 것을 알게 되었다. 또한 관측치들의 plot 을 사전에 그려보거나 차원이 커 그릴 수 없는 경우 SVM 모형에서 적용할 kernel 를 여러가지로 고려하여 더 적합한 모형을 찾을 수 있다. 데이터에 포함된 관측치 개수 n 보다 설명변수의 개수 p 가 더 큰 경우에는 linear kernel 만으로도 training set 과 test set 에서 모두 성능이 뛰어난 모형을 적합할 수 있다. 그러나 많은 경우 n 이 더 크므로 radial 또는 polynomial kernel 을 시도해볼 수 있다.

[Appendix] R code

Exercise 9.4

```
myggplot <- function(...) {
  ggplot(...) + theme_bw() + theme(plot.title = element_text(hjust = 0.5)) +   scale_color_brewer(palette = 'Set1')
}

# Generate a simulated 2-class data set with 100 observations and 2 features.
set.seed(4)
mydata <- data.frame(matrix(rnorm(100*2), ncol = 2)) %>%
  mutate(y = as.factor(ifelse(X1^2 + X2^2 < 1.5, 1, 2)))

myggplot(mydata, legend.name = 'class') + geom_point(aes(X1, X2, color = y, shape = y))
table(mydata$y)

# show that a SVM with a polynomial kernel or a radial kernel will outperform a classifier.
set.seed(4)
train <- sample(100, 50)

svm.linear <- svm(y ~ ., mydata[train,], kernel = 'linear', cost = 10, gamma = 1)
summary(svm.linear)
plot(svm.linear, mydata[train,])
table(true = mydata[train,'y'], pred = svm.linear$fitted)
mean(mydata[train,'y'] != svm.linear$fitted) # train error = 20%

svm.radial <- svm(y ~ ., mydata[train,], kernel = 'radial', cost = 10, gamma = 1)
summary(svm.radial)
plot(svm.radial, mydata[train,])
table(true = mydata[train,'y'], pred = svm.radial$fitted)
mean(mydata[train,'y'] != svm.radial$fitted) # train error = 0%

svm.poly <- svm(y ~ ., mydata[train,], kernel = 'polynomial', cost = 10, gamma = 1)
summary(svm.poly)
plot(svm.poly, mydata[train,])
table(true = mydata[train,'y'], pred = svm.poly$fitted)
mean(mydata[train,'y'] != svm.poly$fitted) # train error = 34%

pred.train <- data.frame(linear = svm.linear$fitted, radial = svm.radial$fitted, poly = svm.poly$fitted)
apply(pred.train, 2, function(pred) mean(mydata[train,'y'] != pred))

pred.test <- data.frame(
  linear = predict(svm.linear, mydata[-train,]),
  radial = predict(svm.radial, mydata[-train,]),
  poly = predict(svm.poly, mydata[-train,])
)
```

```
apply(pred.test, 2, function(pred) mean(mydata[-train,'y'] != pred))
```

Exercise 9.5

```
# (a) Generate a data set with n=500, p=2,
set.seed(500)
mydata <- data.frame(X1 = runif(500) - 0.5, X2 = runif(500) - 0.5) %>% mutate(y = as.factor(1*(X1^2 - X2^2 > 0)))
# (b) Plot
myggplot(mydata) + geom_point(aes(X1, X2, col = y, shape = y))

# (c) Logistic Regression
summary(myglm <- glm(y ~ ., mydata, family = binomial))
# (d) Plot
pred.glm <- as.factor(as.numeric(predict(myglm, type = 'response') > 0.5))
myggplot(mydata) + geom_point(aes(X1, X2, col = pred.glm, shape = y))

# (e)
summary(myglm.poly <- glm(y ~ poly(X1, 2) + poly(X2, 2), mydata, family = binomial))
# (f)
pred.glm <- as.factor(as.numeric(predict(myglm.poly, type = 'response') > 0.5))
myggplot(mydata) + geom_point(aes(X1, X2, col = pred.glm, shape = y))

# (g)
summary(mysvm.linear <- svm(y ~ ., mydata, kernel = 'linear'))
myggplot(mydata, legend.name = 'linear') + geom_point(aes(X1, X2, col = predict(mysvm.linear), shape = y))

# (h)
summary(mysvm.poly <- svm(y ~ ., mydata, kernel = 'polynomial'))
myggplot(mydata, legend.name = 'polynomial') + geom_point(aes(X1, X2, col = predict(mysvm.poly), shape = y))
summary(mysvm.radial <- svm(y ~ ., mydata, kernel = 'radial'))
myggplot(mydata, legend.name = 'radial') + geom_point(aes(X1, X2, col = predict(mysvm.radial), shape = y))

# (i)
pred.table <- data.frame(
  glm.linear = as.factor(as.numeric(predict(myglm, type = 'response') > 0.5)),
  glm.poly = as.factor(as.numeric(predict(myglm.poly, type = 'response') > 0.5)),
  svm.linear = predict(mysvm.linear),
  svm.poly = predict(mysvm.poly),
  svm.radial = predict(mysvm.radial)
)
apply(pred.table, 2, function(pred) mean(mydata$y != pred))
```

Exercise 9.8

```
data(OJ)
# (a)
set.seed(8)
train <- sample(nrow(OJ), 800)

# (b) Fit a support vector classifier using cost=0.01
summary(svm.linear <- svm(Purchase ~ ., OJ[train,], kernel = 'linear', cost = 0.01))

# (c) training and test error rates
misclassification <- function(svm.model) {
  out1 <- table(true = OJ[train,'Purchase'], predicted.train = predict(svm.model, OJ[train,]))
  out2 <- mean(OJ[train,'Purchase'] != predict(svm.model, OJ[train,]))
  out3 <- table(true = OJ[-train,'Purchase'], predicted.test = predict(svm.model, OJ[-train,]))
  out4 <- mean(OJ[-train,'Purchase'] != predict(svm.model, OJ[-train,]))
  return(list(train.mat = out1, train.error = out2, test.mat = out3, test.error = out4))
```

```
}
misclassification(svm.linear)

# (d) tune to select an optimal cost
set.seed(8)
(tune.cost <- tune(svm, Purchase ~ ., data = OJ[train,], kernel = 'linear', ranges = list(cost = c(0.01, seq(0.1, 10, by = 0.5)))))
summary(tune.cost)
plot(tune.cost$performances[,1:2], type = 'l', pch = 19, lwd = 2)
abline(h = min(tune.cost$performance[,2]), lty = 2, col = 'blue')
# (e) predict using new value for cost
best.linear <- tune.cost$best.model
misclassification(best.linear)

# (f) repeat using radial kernel
summary(svm.radial <- svm(Purchase ~ ., OJ[train,], kernel = 'radial', cost = 0.01))
misclassification(svm.radial)

set.seed(8)
(tune.cost.rad <- tune(svm, Purchase ~ ., data = OJ[train,], kernel = 'radial', ranges = list(cost = c(0.01, seq(0.1, 10, by = 0.5)))))
summary(tune.cost.rad)
plot(tune.cost.rad$performances[,1:2], type = 'l', pch = 19, lwd = 2)
best.radial <- tune.cost.rad$best.model
misclassification(best.radial)

# (g) repat using polynomial kernel with degree=2
summary(svm.poly <- svm(Purchase ~ ., OJ[train,], kernel = 'polynomial', degree = 2, cost = 0.01))
misclassification(svm.poly)

set.seed(8)
(tune.cost.poly <- tune(svm, Purchase ~ ., data = OJ[train,], kernel = 'polynomial', degree = 2, ranges = list(cost = c(0.01, seq(0.1, 10, by = 0.5)))))
summary(tune.cost.poly)
plot(tune.cost.poly$performances[,1:2], type = 'l', pch = 19, lwd = 2)
best.poly <- tune.cost.poly$best.model
misclassification(best.poly)

# (h)
model.list <- list(svm.linear, svm.radial, svm.poly, best.linear, best.radial, best.poly)
data.frame(row.names = c('linear','radial','poly','tuned.liniear','tuned.radial','tuned.poly'),
   train = unlist(lapply(model.list, function(obj) misclassification(obj)[[2]])),
   test = unlist(lapply(model.list, function(obj) misclassification(obj)[[4]]))
)
```