Yonsei University, Department of Applied Statistics

Chapter 9 실습

12기 YBIGTA 신보현

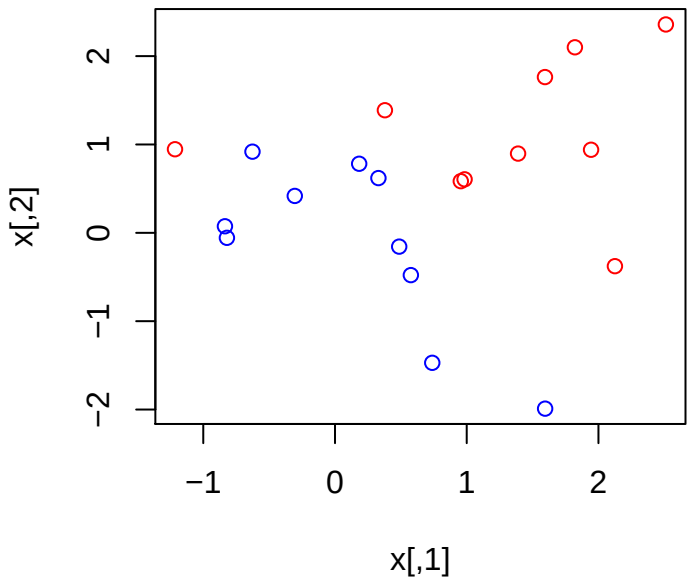
March 2, 2019

e1071 library을 사용하여 support vector classifier와 SVM을 실행해본다.

1. Support Vector Classifier

e1071 library의 svm() 함수를 이용하여 kernel="linear" 옵션을 통해 support vector classifier을 실생할 수 있다. cost 옵션을 통해 margin에 대한 위반의 정도를 정할 수 있다. 만약 작다면 margin은 넓고 많은 support vectors가 margin에 있거나 margin을 위반할 것이다. 만약 크다면 margin이 좁아져서 margin에 있거나 margin을 위반하는 support vectors가 적을 것이다.
이제 svm() 함수를 이용해서 support vector classifier을 적합하자. 우선 예시에서는 2차원 예제로 진행하는데, 결과로 나오는 결정 바운더리를 그리기 위함이다. 두 개의 클래스에 속하는 관측치를 생성하고 클래스들이 선형적으로 분리될 수 있는지 확인한다.

```
knitr::opts_chunk$set(comment=NA, fig.width=4, fig.height=4,fig.align='center',message=FALSE)
set.seed(1)
x=matrix(rnorm(20*2),ncol=2)
y = c(rep(-1,10),rep(1,10))
x[y==1,] = x[y==1,] + 1
plot(x, col=3-y)
```

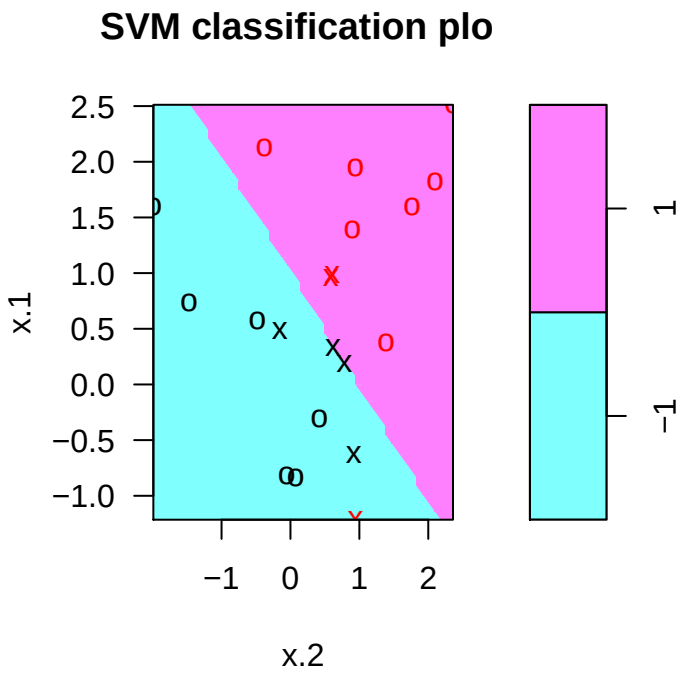


선형 바운더리로 분류될 수 없음을 확인할 수 있다. 따라서 완벽히 클래스를 분류하는 maximal margin classifier가 아니라 어느 정도 오차를 허용하는 support vector classifier로 분류를 해야할 것이다. 여기서 주의할 점은, 반응변수가 factor 변수로 코딩이 되어있어야 한다는 점이다.

```
data = data.frame(x=x,y=as.factor(y))
library(e1071)
fit.svm = svm(y~., data=data, kernel="linear", cost=10, scale=FALSE)
```

여기서는 scale=FALSE로 했지만 상황에 따라서는 scale을 해야할 상황도 있다.
아래와 같이 그림으로 나타낼 수도 있다. 이 때, plot은 입력값으로 적합한 모델과 적합할 때 사용한 데이터임을 주의하자(일반적인 x,y가 아니다)

```
plot(fit.svm,data)
```



support vector classifier을 적합했으므로 결정 바운더리가 선형이다. support vectors는 x로, 그 이외는 o로 표시되었다.
아래와 같이 support vectors에 대한 인덱스를 뽑을 수도 있다.

```
names(fit.svm)
```

```
 [1] "call"         "type"         "kernel"
 [4] "cost"         "degree"       "gamma"
 [7] "coef0"        "nu"           "epsilon"
[10] "sparse"       "scaled"       "x.scale"
```

```
[13] "y.scale"         "nclasses"        "levels"
[16] "tot.nSV"         "nSV"             "labels"
[19] "SV"              "index"           "rho"
[22] "compprob"        "probA"           "probB"
[25] "sigma"           "coefs"           "na.action"
[28] "fitted"          "decision.values" "terms"
```

```
fit.svm$index
```

```
[1]  1  2  5  7 14 16 17
```

```
summary(fit.svm)
```

```

Call:
svm(formula = y ~ ., data = data, kernel = "linear", cost = 10,
    scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10
      gamma:  0.5

Number of Support Vectors:  7

 ( 4 3 )


Number of Classes:  2

Levels:
 -1 1
```

위의 요약으로부터 linear kernel, 즉 support vector classifier가 적합되었고 7개의 support vectors가 있음을 알 수

있다.
만약 작은 값의 cost 옵션을 사용하면 어떻게 될까?

```
fit.svm = svm(y~., data=data, kernel="linear", cost=0.1, scale=FALSE)
summary(fit.svm)
```

```

Call:
svm(formula = y ~ ., data = data, kernel = "linear", cost = 0.1,
    scale = FALSE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.5

Number of Support Vectors:  16

 ( 8 8 )


Number of Classes:  2

Levels:
 -1 1
```

```
fit.svm$index
```

```
 [1]  1  2  3  4  5  7  9 10 12 13 14 15 16 17 18 20
```

margin이 넓어짐에 따라, support vectors가 많아짐을 확인할 수 있다.
안타깝게도, svm() 함수는 선형 결정 바운더리의 추정된 계수나 margin의 폭을 알려주지는 않는다.
e1071 library는 CV를 수행하는 tune() 함수가 있다. 기본값으로 tune() 함수는 10 fold CV를 실행한다. 이 함수를 사용하기 위해서는 고려하고 있는 모델에 대한 관련된 정보를 집어 넣어야 한다. 아래는 linear kernel로 SVM를 실행하기 전, 모수 cost에 대해서 CV를 실행하는 코드이다.

```r
set.seed(1)
tune.out = tune(svm, y~., data=data,kernel="linear",ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)))
summary(tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.1

- Detailed performance results:
   cost error dispersion
1 1e-03  0.70  0.4216370
2 1e-02  0.70  0.4216370
3 1e-01  0.10  0.2108185
4 1e+00  0.15  0.2415229
5 5e+00  0.15  0.2415229
6 1e+01  0.15  0.2415229
7 1e+02  0.15  0.2415229
```

결과를 통해서 최적(가장 낮은 CV error)의 파라미터는 0.1임을 확인할 수 있다. tune() 함수는 최적의 모델을 저장하는데, 아래와 같이 접근한다.

```r
names(tune.out)
```

```
[1] "best.parameters"  "best.performance" "method"
[4] "nparcomb"         "train.ind"        "sampling"
[7] "performances"     "best.model"
```

```r
best.mod = tune.out$best.model
summary(best.mod)
```

```
Call:
best.tune(method = svm, train.x = y ~ ., data = data, ranges = list(cost = c(0.001,
    0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.1
      gamma:  0.5

Number of Support Vectors:  16

 ( 8 8 )


Number of Classes:  2

Levels:
 -1 1
```

predict() 함수는 테스트 관측치의 클래스를 예측하기 위해 쓰인다. 우선 테스트 관측치를 생성하자.

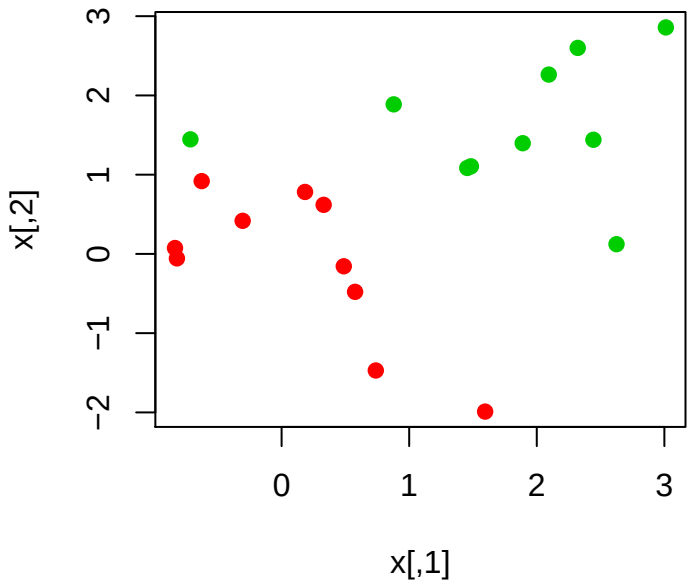
```
set.seed(1)
xtest = matrix(rnorm(20*2),ncol=2)
ytest = sample(c(-1,1),20,rep=TRUE)
xtest[ytest==1,] = xtest[ytest==1,] + 1
testdat = data.frame(x=xtest,y=as.factor(ytest))
```

이제 tune() 함수를 통해서 얻은 최적의 모델을 이용하여 클래스를 예측해보자.

```
svm.pred = predict(best.mod,testdata)

Error in predict.svm(best.mod, testdata): 객체 'testdata'를 찾을 수 없습니다

table(svm.pred,ytest)

Error in table(svm.pred, ytest): 객체 'svm.pred'를 찾을 수 없습니다
```

이제 두 개의 클래스가 선형으로 분리될 수 있는 상황을 생각해보자. 이러한 상황에서는 두 클래스를 완전히 분리하는 separating hyperplane을 찾을 수 있을 것이다. 우선 두 클래스가 완전히 분리되도록 데이터를 생성하자.

```
x[y==1,] = x[y==1,]+0.5
plot(x,col=(y+5)/2,pch=19)
```

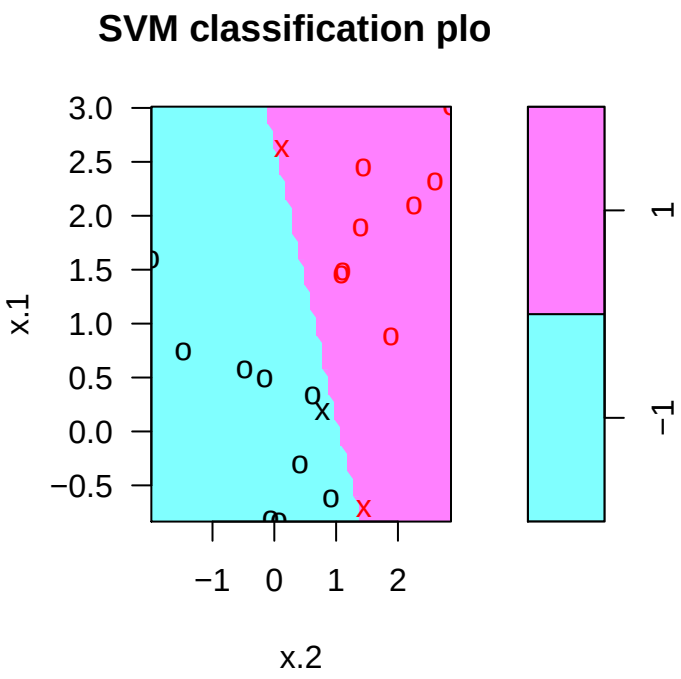


두 클래스가 선형으로 분리될 수 있다. 이제 support vector classifier을 적합하는데 cost 옵션을 크게 함으로써 모든 관측치가 잘 분류되도록 해보자.

```
data = data.frame(x=x,y=as.factor(y))
fit.svm = svm(y~.,data=data,kernel="linear",cost=1e5)
summary(fit.svm)
```

```

Call:
svm(formula = y ~ ., data = data, kernel = "linear", cost = 1e+05)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
```

```
       cost:  1e+05
      gamma:  0.5

Number of Support Vectors:  3

 ( 1 2 )


Number of Classes:  2

Levels:
 -1 1

plot(fit.svm,data)
```



모든 관측치가 완벽히 분리되었고 단 3개의 support vectors가 쓰였다. 하지만 결정 바운더리 근처에 support vectors가 아닌 o로 표시된 관측치가 굉장히 가까이 있는 것으로 보아 margin이 매우 좁음을 유추할 수 있다. 이는 훈련 데이터에 과적합된 상태로 테스트 데이터에는 매우 좋지 않은 결과를 낼 것이다.
실제로 tune() 함수를 통해서 확인 결과, 다른 파라미터가 더 좋은 성능을 냄을 확인할 수 있었다.

```
tune.out = tune(svm, y~., data=data,kernel="linear",ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100,1e3,1e
summary(tune.out)
```

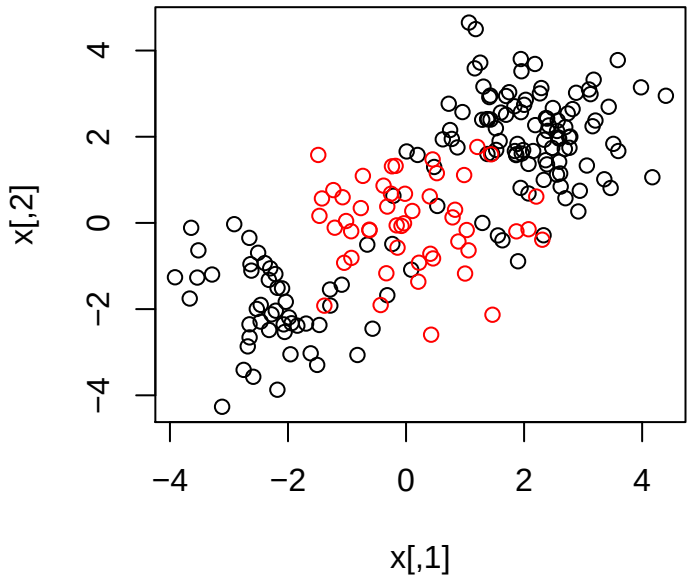
```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
  0.1

- best performance: 0.05

- Detailed performance results:
    cost error dispersion
1  1e-03  0.25  0.4249183
2  1e-02  0.25  0.4249183
3  1e-01  0.05  0.1581139
4  1e+00  0.05  0.1581139
5  5e+00  0.10  0.2108185
6  1e+01  0.10  0.2108185
7  1e+02  0.10  0.2108185
8  1e+03  0.10  0.2108185
9  1e+04  0.10  0.2108185
10 1e+05  0.10  0.2108185
```

2. Support Vector Machine

SVM을 적합하기 위해서 동일한 svm() 함수를 사용하는데 kernel 옵션만 달리 설정한다. polynomial kernel 사용시에는 degree 옵션을 이용하여 차수를 설정한다. 그리고 radial kernel 사용시에는 gamma 옵션을 통해서 γ 을 설정한다.
우선 비선형 클래스 바운더리를 가지는 데이터를 생성하자.

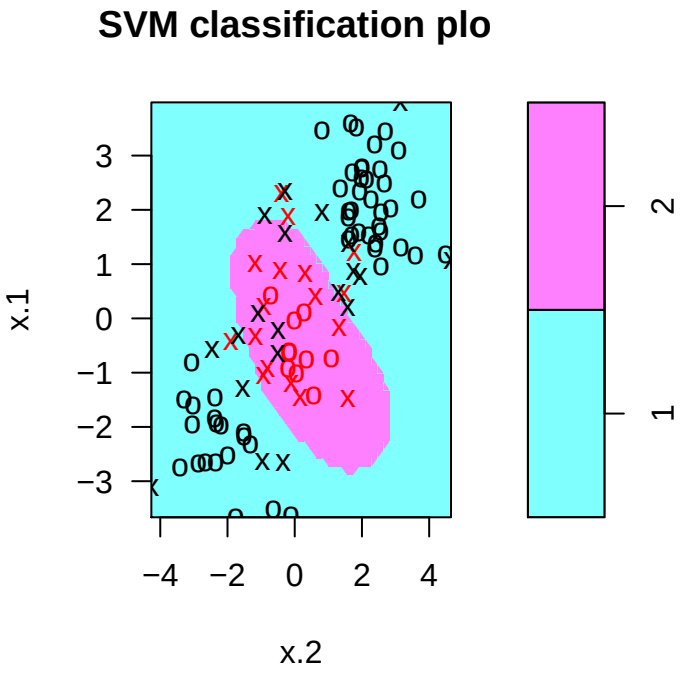
```
set.seed(1)
x = matrix(rnorm(200*2),ncol=2)
x[1:100,] = x[1:100,] + 2
```

```r
x[101:150,] = x[101:150,]-2
y = c(rep(1,150),rep(2,50))
data = data.frame(x=x, y=as.factor(y))
plot(x,col=y)
```



훈련 데이터와 테스트 데이터로 나누고 gamma=1로 radial kernel을 사용하여 svm을 적합해본다.

```r
train = sample(1:nrow(data),nrow(data)/2)
radial.fit = svm(y~., data=data[train,], kernel="radial", gamma=1, cost=1)
plot(radial.fit,data[train,])
```



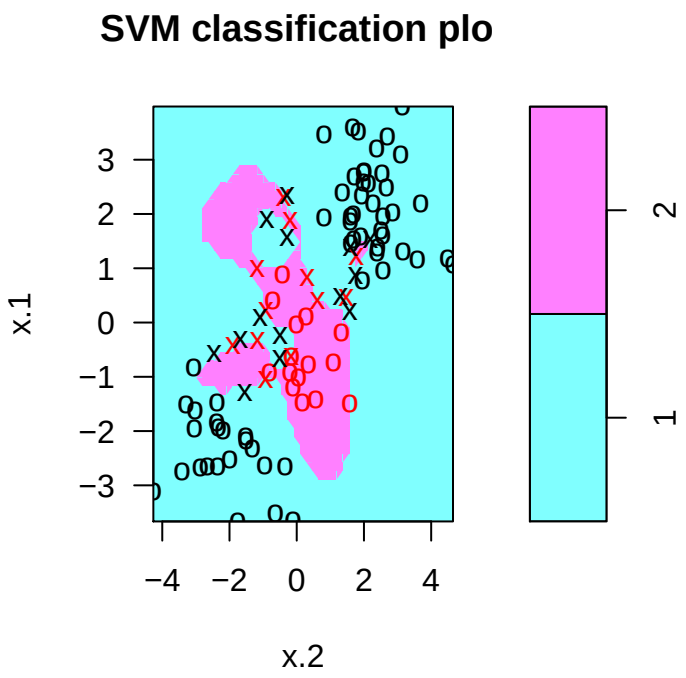

```
summary(radial.fit)
```

```
Call:
svm(formula = y ~ ., data = data[train, ], kernel = "radial",
    gamma = 1, cost = 1)


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

Number of Support Vectors:  37

 ( 17 20 )


Number of Classes:  2
```

```
Levels:
 1 2
```

만약 cost을 더 늘린다면 결정 바운더리는 어떤 모양이 될까? margin이 좁아져서 결정 바운더리가 매우 불규칙적인 모양으로 해당 훈련 데이터에는 매우 잘 적합되겠지만 테스트 데이터에 대해서는 매우 좋지 않은 성능을 보일 것으로 유추할 수 있다.

```
radial.fit = svm(y~., data=data[train,], kernel="radial", gamma=1, cost=1e5)
plot(radial.fit,data[train,])
```



이렇게 임의로 cost와 gamma을 정하는 것보다는 앞서 살펴본 tune() 함수를 이용하여 가장 최적의 파라미터를 찾는 것이 좀 더 객관적인 방법이다.

```
tune.out = tune(svm, y~., data=data[train,],kernel="radial",
ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100,1e3,1e4,1e5),gamma=c(0.5,1,2,3,4)))
tune.out$best.model



Call:
best.tune(method = svm, train.x = y ~ ., data = data[train, ],
```

```
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100, 1000,
        10000, 1e+05), gamma = c(0.5, 1, 2, 3, 4)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.5

Number of Support Vectors:  36
```

가장 최적의 모델은 cost = 1, gamma=0.5으로 확인되었다. 해당 모델로 테스트 데이터에 대해서 예측을 해보자.

```
radial.pred = predict(tune.out$best.model,newdata=data[-train,])
table(prediction=radial.pred,realvalue=data$y[-train])


          realvalue
prediction  1  2
         1 72  7
         2  5 16
```

3. ROC Curves

ROCR 패키지는 그림 9.10, 9.11의 ROC 커브를 그리기 위해 사용된다. 우선 각 관측치에 대한 numerical score을 담고있는 pred, 각 관측치의 클래스 라벨을 담고있는 truth가 주어질 때, ROC 커브를 그리는 함수를 작성하자.

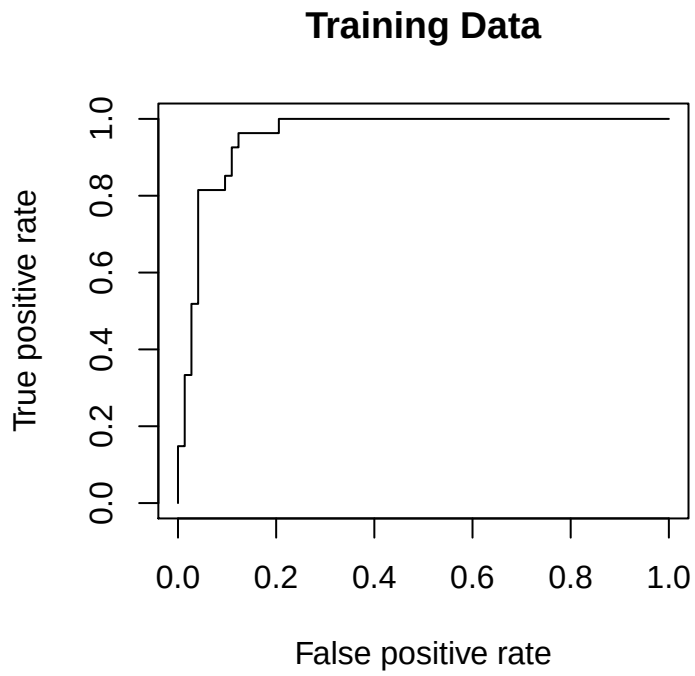
```
library(ROCR)
rocplot = function(pred, truth, ...){
predob = prediction(pred, truth)
perf = performance(predob, "tpr","fpr")
plot(perf,...)
}
```

https://cran.r-project.org/web/packages/ROCR/ROCR.pdf을 참조하면, performance는 Function to create performance objects이다. 여기서 measure='tpr', x.measure='fpr'로 설정을 하면 ROC 커브를 만들어주는 듯 하다. prediction 함수는 Function to create prediciton objects이다. 입력 값으로 prediction 값의 벡터, 실제 label 벡터

를 받는다. 이를 통해서 prediction object을 생성하는데, 이것을 후에 performance 함수의 첫 번째 argument에 입력한다.

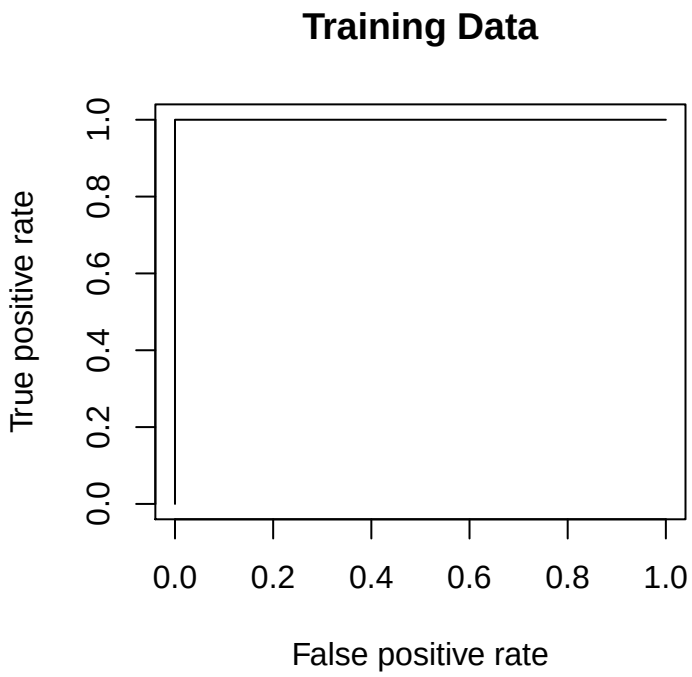
위에서 만든 rocplot에서 pred 값은 클래스 라벨이 아니라 각 관측치에 대한 numerical score이다. 이를 svm에서 어떻게 구할까? svm() 함수를 실행하여 각 관측치에 대해서 fitted values을 얻을 수 있다. 즉, support vector classifier에 대해서 어떤 관측치 $X = (X_1, \cdots, X_p)^T$의 fitted value는 $\hat{\beta}_0 + \hat{\beta}_1 X_1 + \cdots + \hat{\beta}_p X_p$의 형태이고 non-linear kernel의 SVM은 $\beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(x, x_i)$의 형태이다. 실질적으로, 이 fitted value의 부호가 해당 관측치의 클래스 라벨을 결정하는 것이다. 0보다 크면 클래스 A에, 그렇지 않으면 다른 클래스에 할당한다. SVM 적합에서 fitted value을 얻고 싶으면 decision.values=TRUE나는 옵션을 추가해야 한다.

```
svm.fit_0.5 = svm(y~., data=data[train,],kernel="radial",cost=1,gamma=0.5,decision.values=TRUE)
rocplot(svm.fit_0.5$decision.values,data[train,'y'],main='Training Data')
```



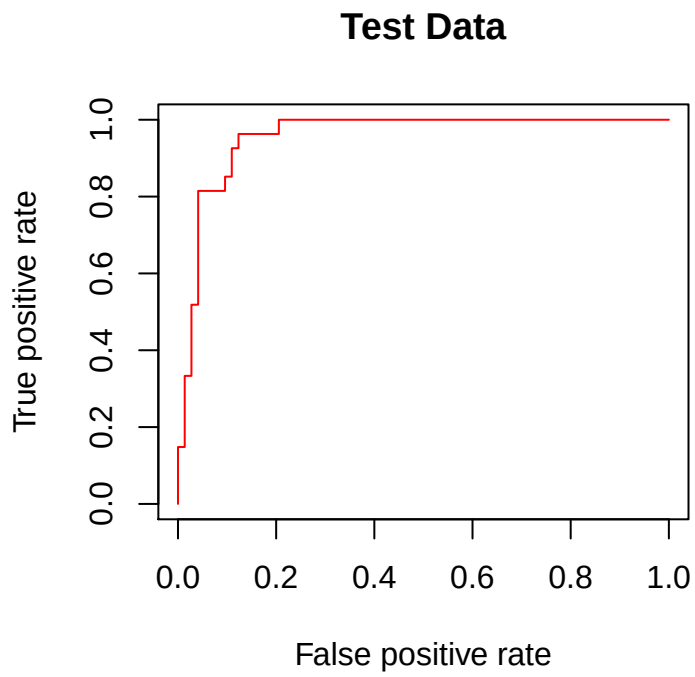
꽤정확한 예측을 하는 것으로 보인다. 이제 γ을 높여서 flexible한 적합을 통해 정확도를 향상시켜보자.

```
svm.fit_50 = svm(y~., data=data[train,],kernel="radial",cost=1,gamma=50,decision.values=TRUE)
rocplot(svm.fit_50$decision.values,data[train,'y'],main='Training Data')
```

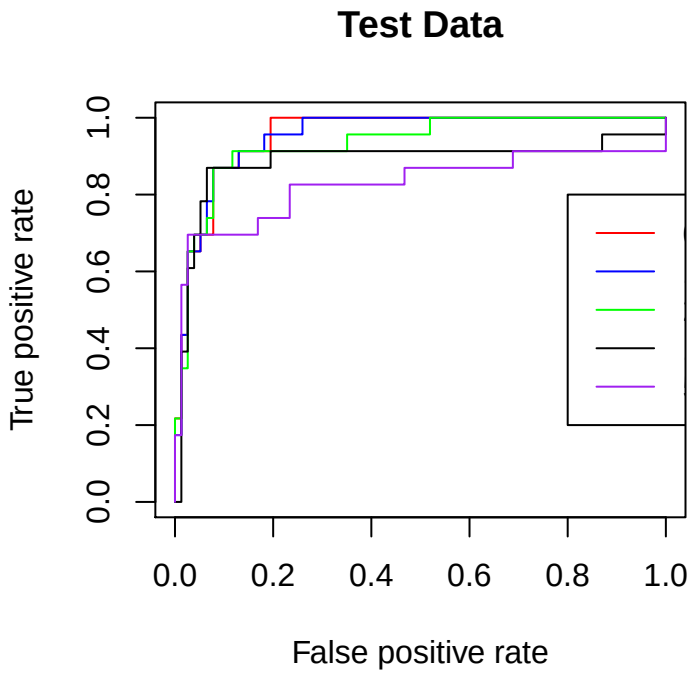


ROC 커브가 좌상단에 거의 인접해있다. 하지만 이는 어디까지나 훈련 데이터에 대한 ROC 커브이므로 이를 맹신해서는 안 된다. 테스트 데이터에 대해서 $\gamma = 0.5, 1, 2, 10, 50$일 때 ROC 커브를 그려보자.

```
svm.fit_1 = svm(y~., data=data[train,],kernel="radial",cost=1,gamma=1,decision.values=TRUE)
svm.fit_2 = svm(y~., data=data[train,],kernel="radial",cost=1,gamma=2,decision.values=TRUE)
svm.fit_10 = svm(y~., data=data[train,],kernel="radial",cost=1,gamma=10,decision.values=TRUE)
rocplot(svm.fit_0.5$decision.values,data[train,'y'],main='Test Data',col='red')
```




```
fitted_0.5 = attributes(predict(svm.fit_0.5,data[-train,],decision.values=T))$decision.values
fitted_1 = attributes(predict(svm.fit_1,data[-train,],decision.values=T))$decision.values
fitted_2 = attributes(predict(svm.fit_2,data[-train,],decision.values=T))$decision.values
fitted_10 = attributes(predict(svm.fit_10,data[-train,],decision.values=T))$decision.values
fitted_50 = attributes(predict(svm.fit_50,data[-train,],decision.values=T))$decision.values
rocplot(fitted_0.5,data[-train,'y'],main='Test Data',col='red')
rocplot(fitted_1,data[-train,'y'],add=T,col='blue')
rocplot(fitted_2,data[-train,'y'],add=T,col='green')
rocplot(fitted_10,data[-train,'y'],add=T,col='black')
rocplot(fitted_50,data[-train,'y'],add=T,col='purple',smooth=T)
legend(0.8,0.8,
legend=c('0.5','1','2','10','50'),
col = c('red','blue','green','black','purple'),
lty = c(1,1,1,1,1,1))
```

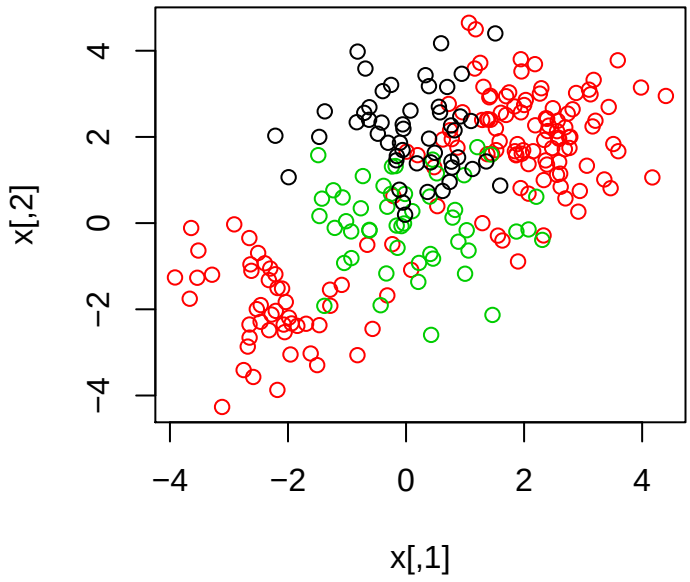


테스트 데이터에 대해서 γ가 1또는 2인 SVM의 ROC 커브가 좌상단에 가장 가까운듯하다.

4. SVM with Multiple Classes

만약 반응변수가 두 개 이상의 클래스를 포함하면, svm() 함수는 one-versus-one 접근법을 통해서 SVM을 실행한다. 여기서는 세 번째 클래스를 만들어서 이를 살펴보자.

```
set.seed(1)
x = rbind(x, matrix(rnorm(50*2), ncol=2))
y = c(y, rep(0,50))
x[y==0,2] = x[y==0,2]+2
data = data.frame(x=x,y=as.factor(y))
plot(x,col=y+1)
```




```
svm.fit = svm(y~., data=data, kernel='radial', cost=10, gamma=1)
plot(svm.fit,data)
```