Mingtao_Gao_HW6

Mingtao Gao

3/2/2020

```r
# Libraries that will be used for this homework
library(patchwork)
library(rcfss)
library(e1071)
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

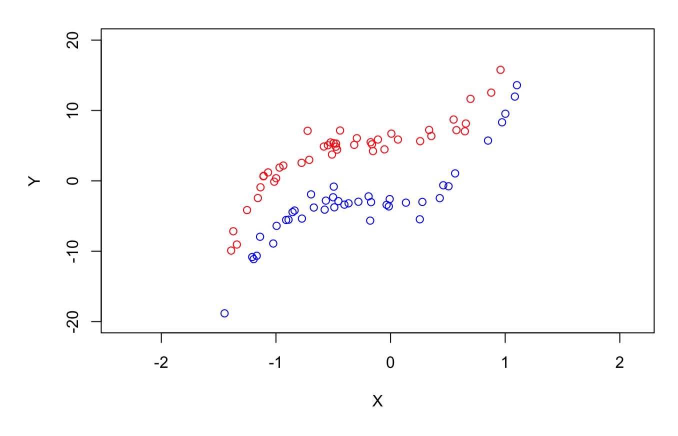
```r
library(kernlab)
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
##
##     alpha
```

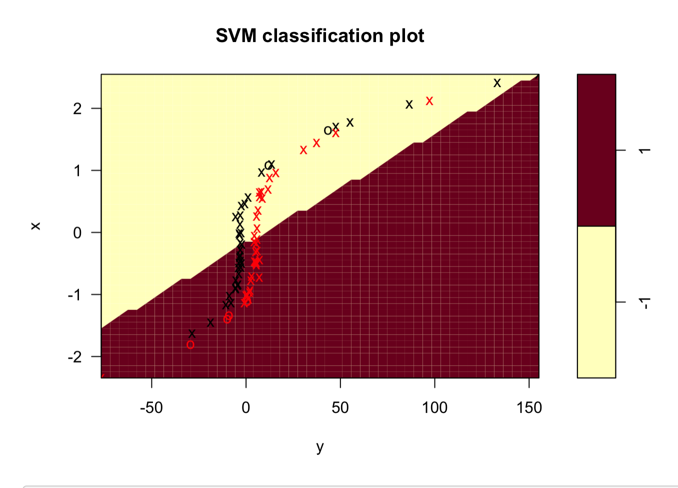
Non-Linear Separation 1.

```r
set.seed(1234)
# Create non-linear relation dataset
X <- rnorm(100)
Y <- 8 * X^3 + 4 * X^2 + rnorm(100) + 1

# Create separation
class = sample(100, 50)
Y[class] = Y[class] + 4
Y[-class] = Y[-class] - 4

# Add class data to the dataset
z = rep(-1, 100)
z[class] = 1
data = data.frame(x = X, y = Y, z = as.factor(z))

# Plot the data
plot(X[class], Y[class], col = "red", xlab = "X", ylab = "Y", ylim = c(-20, 20))
points(X[-class], Y[-class], col = "blue")
```



```r
# Split dataset to training and testing
train = sample(100, 80)
data.train = data[train, ]
data.test = data[-train, ]
```

```r
# Support Vector Classifier (Linear Kernel)
svm.linear <- svm(z ~., data=data.train,
                  kernel='linear',
                  scale=FALSE, cost=5)
plot(svm.linear, data=data.train)
```



```r
# Support Vector Classifier (Linear Kernel) -- Training Error
table(predict=predict(svm.linear, data.train), truth=data.train$z)
```

```
##        truth
## predict -1  1
##      -1 18 14
##      1  21 27
```

The support vector classifier with linear kernel made 35 errors on the training data, the error rate is 35/80 = 0.4375.

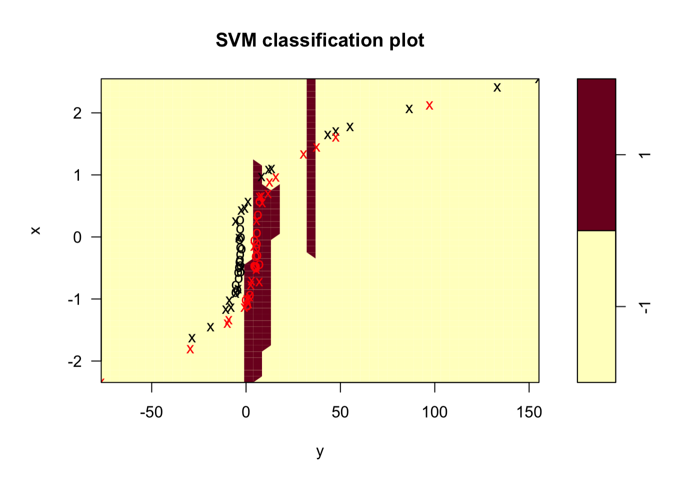
```r
# Support Vector Classifier (Linear Kernel) -- Testing Error
table(predict=predict(svm.linear, data.test), truth=data.test$z)
```

```
##        truth
## predict -1 1
##      -1  5 2
##      1   6 7
```

The support vector classifier with linear kernel made 8 errors on the training data, the error rate is 8/20 = 0.4.

The high error rate of SVC with linear kenel shows that it performed very badly with data that has a non-linear separation between two features.Next, let's move to SVC with a radial kenel and evaluate its performance.

```r
# Support Vector Classifier (Radial Kernel)
svm.radial <- svm(z ~., data=data.train,
                  kernel='radial',
                  scale=FALSE, cost=5)
plot(svm.radial, data=data.train)
```



```r
# Support Vector Classifier (Radial Kernel) -- Training Error
table(predict=predict(svm.radial, data.train), truth=data.train$z)
```

```
##        truth
## predict -1  1
##      -1 36  1
##      1   3 40
```

The support vector classifier with radial kernel only made 4 errors on the training data, the error rate is 4/80 = 0.05.

```r
# Support Vector Classifier (Radial Kernel) -- Testing Error
table(predict=predict(svm.radial, data.test), truth=data.test$z)
```

```
##        truth
## predict -1 1
##      -1  8 3
##      1   3 6
```

The support vector classifier with radial kernel made 6 errors on the training data, the error rate is 6/20 = 0.3.

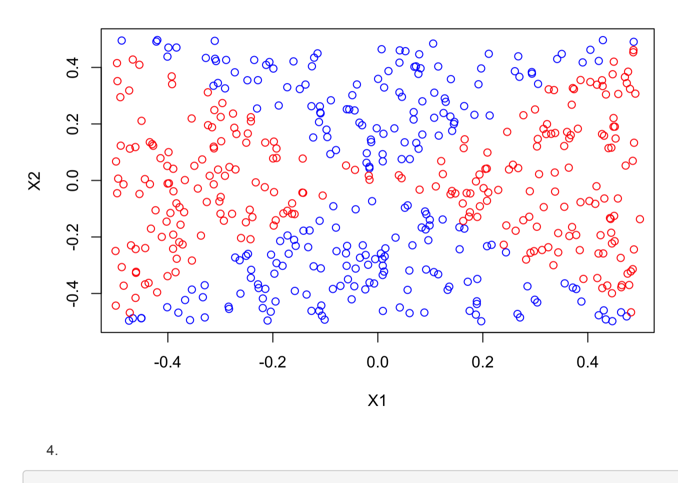
We can see there is a great reduction in training and testing error rate with using radial kernel for suuport vector classifier when there is a non-linear separation between two classes. A support vector machine with a radial kernel indeed outperformed a support vector classifier on the training data in this setting.

SVM vs. Logistic Regression 2.

```r
# Generate dataset
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- 1*(x2^2 - x1^2 > 0)
data = data.frame(x1 = x1, x2 = x2, y = y)
```

3.

```r
# Plot the observations
plot(x1[y == 1], x2[y == 1], col = "blue", xlab = "X1", ylab = "X2")
points(x1[y == 0], x2[y == 0], col = "red")
```



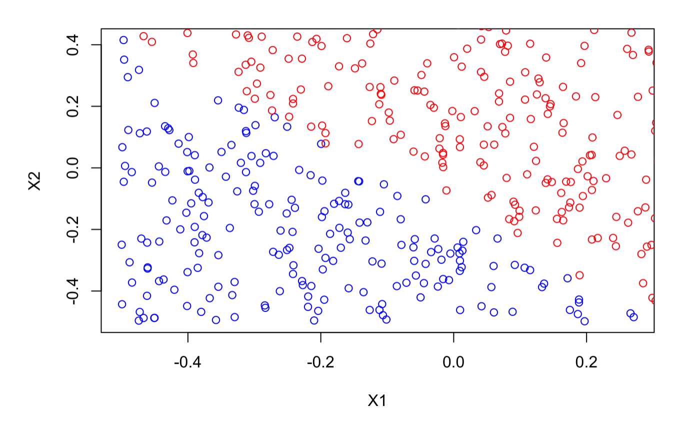
4.

```r
# Fit a logistic regression model with linear relation between x1 and x2
logre.linear = glm(y ~ x1 + x2, family="binomial")
summary(logre.linear)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.296  -1.150  -1.040   1.196   1.342
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.04679    0.08998  -0.520    0.603
## x1          -0.36555    0.30393  -1.203    0.229
## x2          -0.31404    0.31718  -0.990    0.322
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.95  on 499  degrees of freedom
## Residual deviance: 690.41  on 497  degrees of freedom
## AIC: 696.41
##
## Number of Fisher Scoring iterations: 3
```

5.

```r
# Predict the data and plot the predictions
l.prob = predict(logre.linear, newdata=data, type="response")
l.pred = ifelse(l.prob > 0.5, 1, 0)
plot(data[l.pred == 1, ]$x1, data[l.pred == 1, ]$x2,
     xlab = "X1", ylab = "X2", col = "blue")
points(data[l.pred == 0, ]$x1, data[l.pred == 0, ]$x2, col = "red")
```



We can observe the predicted decision boundary looks like a linear relation.

6.

```r
# Fit a logistic regression model with non-linear relation between x1 and x2
logre.nlinear = glm(y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family="binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

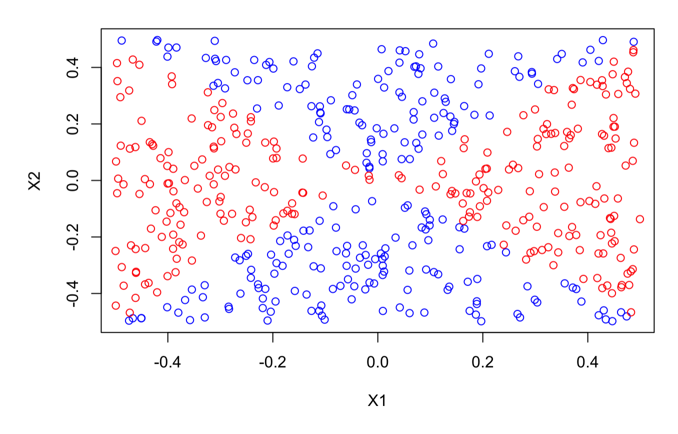
```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
summary(logre.nlinear)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial")
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -1.399e-03  -2.000e-08  -2.000e-08   2.000e-08   1.294e-03
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -104.6     6019.0  -0.017    0.986
## poly(x1, 2)1    -2290.5    85034.3  -0.027    0.979
## poly(x1, 2)2   -33504.2  1008321.0  -0.033    0.973
## poly(x2, 2)1    -2610.8    83247.0  -0.031    0.975
## poly(x2, 2)2    32428.1   953828.0   0.034    0.973
## I(x1 * x2)       -328.1    52168.8  -0.006    0.995
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9295e+02  on 499  degrees of freedom
## Residual deviance: 4.3736e-06  on 494  degrees of freedom
## AIC: 12
##
## Number of Fisher Scoring iterations: 25
```

7.

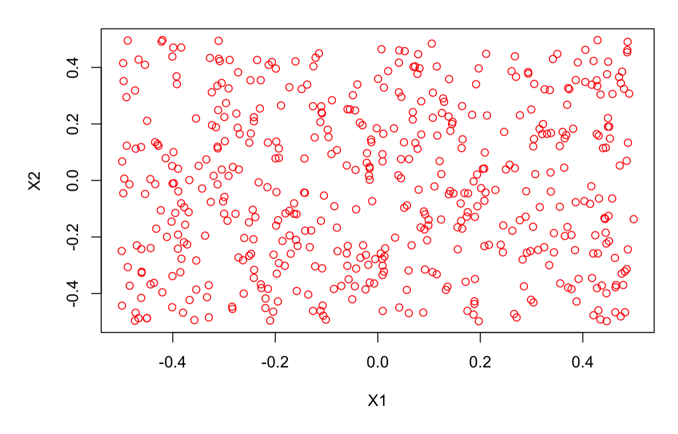
```r
# Predict the data and plot the predictions
nl.prob = predict(logre.nlinear, newdata=data, type="response")
nl.pred = ifelse(nl.prob > 0.5, 1, 0)
plot(data[nl.pred == 1, ]$x1, data[nl.pred == 1, ]$x2,
     xlab = "X1", ylab = "X2", col = "blue")
points(data[nl.pred == 0, ]$x1, data[nl.pred == 0, ]$x2, col = "red")
```



We can see the non-linear logistic regression model performs much better than the linear one and its non-linear decision boundary is very close to the true decision boundary.

8.

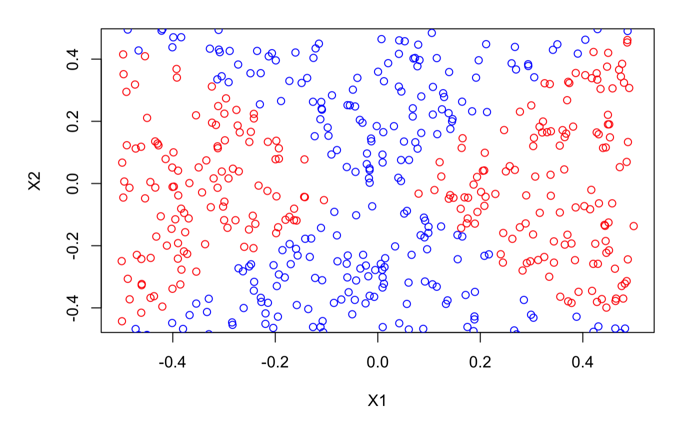
```r
# Fit Support Vector Classifier (linear kernel)
svm.linear <- svm(as.factor(data$y) ~ x1 + x2, data, kernel="linear", cost=0.01)
svml.pred <- predict(svm.linear, data)
plot(data[svml.pred == 0, ]$x1, data[svml.pred == 0, ]$x2,
     xlab = "X1", ylab = "X2", col="red")
points(data[svml.pred == 1, ]$x1, data[svml.pred == 1, ]$x2, col="blue")
```



We can see SVC with linear kernel performed very poorly. It classified all values into one single class.

9.

```r
# Fit Support Vector Classifier (non-linear/radial kernel)
svm.nlinear <- svm(as.factor(data$y) ~ x1 + x2, data, kernel="radial", gamma=1)
svmnl.pred <- predict(svm.nlinear, data)
plot(data[svmnl.pred == 0, ]$x1, data[svmnl.pred == 0, ]$x2,
     xlab = "X1", ylab = "X2", col="red")
points(data[svmnl.pred == 1, ]$x1, data[svmnl.pred == 1, ]$x2, col="blue")
```

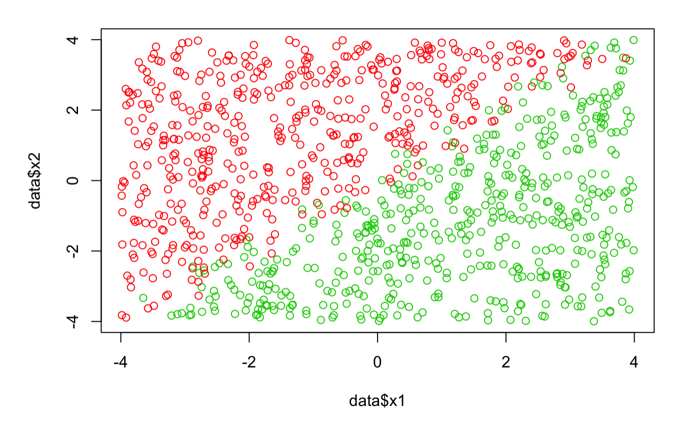


We can see the non-linear kernel also outperform the linear kernel used for support vector classifier.

10. Results and Comments: From the above analysis on linear/non-linear logistic regression and linear/non-linear support vector classifier, we can find that, in this setting with a non-linear decision boundary, SVC with non-linear kernel and logistic regression with interaction terms both perform well on fitting the data. However, SVC with linear kernel and linear logistic regression performed very poorly when the decision boundary is non-linear. The tradeoff for logistic regression is that we need to testing many interactive relations in order to find the best performed model. The tradeoff for SVC is that we also need to tuning the gamma value in order to fit the best model.

Tuning Cost 11.

```r
# Generate barely linear separated dataset
x1 <- runif(1000, -4, 4)
x2 <- runif(1000, -4, 4)
y <- rep(NA, 1000)
for (i in seq(1, 1000)){
  if (x1[i] - x2[i] > 0.5) {
    y[i] <- 1
  } else if (x1[i] - x2[i] < -0.5){
    y[i] <- 0
  } else {
    y[i] <-sample(c(0,1), replace=TRUE, size=1)
  }
}
data <- data.frame(x1 = x1, x2 = x2, y = as.factor(y))
plot(data$x1, data$x2, col=as.integer(data$y) + 1)
```



12.

```r
# Split data into training and testing set
train = sample(100, 80)
data.train = data[train, ]
data.test = data[-train, ]
```

```r
# Tuning cost
costs = c(0.01, 0.1, 1, 5, 10, 100, 1000, 10000)
tune.out <- tune(svm, y ~ ., data=data.train, kernel="linear",
                 ranges=list(cost = costs))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1
##
## - Detailed performance results:
##    cost  error dispersion
## 1 1e-02 0.3125 0.14731391
## 2 1e-01 0.1625 0.15645819
## 3 1e+00 0.1250 0.11785113
## 4 5e+00 0.1125 0.10944938
## 5 1e+01 0.1000 0.09860133
## 6 1e+02 0.1125 0.09223310
## 7 1e+03 0.1125 0.09223310
## 8 1e+04 0.1125 0.09223310
```

The cross-validation errors are displayed above. We can find when the cost value starts to increase, the CV errors starts to decrease and remain stable at a certain point. The best cost value based on CV error rates is 10.

```r
# Compute the training errors with different cost values
train.errs <- rep(NA, length(costs))
for (cost in costs) {
  svm.fit <- svm(y ~ ., data=data.train, kernel='linear', cost=cost)
  res <- table(prediction=predict(svm.fit, newdata=data.train), truth=data.train$y)
  # calculate the training error
  train.errs[match(cost, costs)] <- (res[2,1] + res[1,2]) / sum(res)
}
train.errs
```

```
## [1] 0.2250 0.1125 0.1000 0.1000 0.1000 0.1125 0.0875 0.0875
```

The training errors show a similar trend as the CV error rates. As cost values increase, the training errors also decrease first and then remain at a stable level. The best cost value based on training errors is 1.

13.

```r
# Compute the testing errors with different cost values
test.errs <- rep(NA, length(costs))
for (cost in costs) {
  svm.fit <- svm(y ~ ., data=data.train, kernel='linear', cost=cost)
  res <- table(prediction=predict(svm.fit, newdata=data.test), truth=data.test$y)
  test.errs[match(cost, costs)] <- (res[2,1] + res[1,2]) / sum(res)
}
test.errs
```

```
## [1] 0.31956522 0.09239130 0.06956522 0.05217391 0.05543478 0.05434783 0.05000000
## [8] 0.05000000
```

The testing errors also show a similar trend as training and CV errors. As cost values increase, the testing errors also decrease first and then remain at a stable level. The best cost value based on training errors is 1.

14. When we have dataset that the two classes are just barely linearly separable, the cost value that is either too big or too small would lead to higher traing/testing/CV error rates and poor performance. In general, a small cost creates a large margin and allows more misclassifications, with higher training errors, while a large cost creates a narrow margin and permits fewer misclassifications, with lower training errors. Thus, after reaching a certain level, the increasement in cost will not lead to any changes in error rates for testing/training dataset. For CV, training, and testing error rates across all cost values, when cost is equal to 1, the error rate is minimized.

Predicting attitudes towards racist college professors 15.

```r
# Load the dataset
gss.test <- read.csv('data/gss_test.csv')
gss.train <- read.csv('data/gss_train.csv')
```

```r
# SVC with linear kernel
tune.out <- tune(svm, colrac ~ ., data=gss.train, kernel="linear",
                 ranges=list(cost=c(0.01, 0.1, 1, 5, 10, 100)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.1829205
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-02 0.1829205 0.03724689
## 2 1e-01 0.2001940 0.03940306
## 3 1e+00 0.2017173 0.03974464
## 4 5e+00 0.2018641 0.03992938
## 5 1e+01 0.2019736 0.03996796
## 6 1e+02 0.2056114 0.03919998
```

Based on the CV error rates reported above, when cost is equal to 0.01, the SVC with linear kernel performs the best with the smallest CV error rate. As the cost value increases, the CV error rates started to increase.

16.

```r
# SVC with polynomial kernel
tune.out <- tune(svm, colrac ~ ., data=gss.train, kernel="polynomial",
                 ranges=list(cost=c(0.01, 0.1, 1, 5, 10, 100),
                             degree=c(2, 3, 4)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##     1      3
##
## - best performance: 0.1510476
##
## - Detailed performance results:
##     cost degree     error  dispersion
## 1  1e-02      2 0.4069942 0.030528692
## 2  1e-01      2 0.2642998 0.028310684
## 3  1e+00      2 0.2153774 0.024515812
## 4  5e+00      2 0.2844111 0.029608391
## 5  1e+01      2 0.3421966 0.032831648
## 6  1e+02      2 0.7157320 0.072141129
## 7  1e-02      3 0.3704906 0.030983154
## 8  1e-01      3 0.1794833 0.017655565
## 9  1e+00      3 0.1510476 0.014284275
## 10 5e+00      3 0.1601927 0.016197828
## 11 1e+01      3 0.1629656 0.016315396
## 12 1e+02      3 0.1656145 0.016514193
## 13 1e-02      4 0.4158648 0.030665112
## 14 1e-01      4 0.3121320 0.031169494
## 15 1e+00      4 0.1869533 0.011567064
## 16 5e+00      4 0.1864760 0.010511355
## 17 1e+01      4 0.1885236 0.009493718
## 18 1e+02      4 0.1890164 0.009212695
```

Based on the CV error rates reported above, when cost is equal to 1 and degree is equal to 3, the SVC with polynomial kernel performs the best with the smallest CV error rate.

```r
# SVC with radial kernel
tune.out <- tune(svm, colrac ~ ., data=gss.train, kernel="radial",
                 ranges=list(cost=c(0.01, 0.1, 1, 5, 10, 100),
                             gamma=c(0.01, 0.1, 1, 5, 10, 100)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1  0.01
##
## - best performance: 0.148982
##
## - Detailed performance results:
##     cost gamma     error  dispersion
## 1  1e-02 1e-02 0.2814061 0.034625803
## 2  1e-01 1e-02 0.1552009 0.011593310
## 3  1e+00 1e-02 0.1489820 0.012883444
## 4  5e+00 1e-02 0.1571259 0.011694000
## 5  1e+01 1e-02 0.1659180 0.009393847
## 6  1e+02 1e-02 0.1766483 0.012079176
## 7  1e-02 1e-01 0.4246215 0.046278035
## 8  1e-01 1e-01 0.3808932 0.041308834
## 9  1e+00 1e-01 0.2163368 0.005895547
## 10 5e+00 1e-01 0.2170002 0.005731608
## 11 1e+01 1e-01 0.2170002 0.005731608
## 12 1e+02 1e-01 0.2170002 0.005731608
## 13 1e-02 1e+00 0.4259575 0.046464327
## 14 1e-01 1e+00 0.3935507 0.042926963
## 15 1e+00 1e+00 0.2498751 0.002203633
## 16 5e+00 1e+00 0.2498751 0.002203633
## 17 1e+01 1e+00 0.2498751 0.002203633
## 18 1e+02 1e+00 0.2498751 0.002203633
## 19 1e-02 5e+00 0.4259575 0.046464329
## 20 1e-01 5e+00 0.3935510 0.042926982
## 21 1e+00 5e+00 0.2498769 0.002203960
## 22 5e+00 5e+00 0.2498769 0.002203960
## 23 1e+01 5e+00 0.2498769 0.002203960
## 24 1e+02 5e+00 0.2498769 0.002203960
## 25 1e-02 1e+01 0.4259575 0.046464329
## 26 1e-01 1e+01 0.3935510 0.042926982
## 27 1e+00 1e+01 0.2498769 0.002203960
## 28 5e+00 1e+01 0.2498769 0.002203960
## 29 1e+01 1e+01 0.2498769 0.002203960
## 30 1e+02 1e+01 0.2498769 0.002203960
## 31 1e-02 1e+02 0.4259575 0.046464329
## 32 1e-01 1e+02 0.3935510 0.042926982
## 33 1e+00 1e+02 0.2498769 0.002203960
## 34 5e+00 1e+02 0.2498769 0.002203960
## 35 1e+01 1e+02 0.2498769 0.002203960
## 36 1e+02 1e+02 0.2498769 0.002203960
```

Based on the CV error rates reported above, when cost is equal to 1 and gamma is equal to 0.01, the SVC with polynomial kernel performs the best with the smallest CV error rate.

```r
# Prediction - SVC (linear)
svm.linear <- svm(colrac ~., data=gss.train, kernel='linear', cost=0.01)
table(predict=ifelse(predict(svm.linear, gss.test) > 0.5, 1, 0),
      truth=gss.test$colrac)
```

```
##        truth
## predict   0   1
##       0 117  27
##       1 111 238
```

```r
# Prediction - SVC (polynomial)
svm.poly <- svm(colrac ~., data=gss.train, kernel='polynomial',
                cost=1, degree=3)
table(predict=ifelse(predict(svm.poly, gss.test) > 0.5, 1, 0),
      truth=gss.test$colrac)
```

```
##        truth
## predict   0   1
##       0 153  45
##       1  75 220
```

```r
# Prediction - SVC (radial)
svm.radial <- svm(colrac ~., data=gss.train, kernel='radial',
                cost=1, gamma=0.01)
table(predict=ifelse(predict(svm.radial, gss.test) > 0.5, 1, 0),
      truth=gss.test$colrac)
```

```
##        truth
## predict   0   1
##       0 154  34
##       1  74 231
```

We found that among all three models, the radial kernel SVC with tuned cost and gamma value outperformed the others, which shows a non-linear decision boundary between the two classes.