Support Vector Machines

Anuraag Girdhar

```
library(e1071)
library(tidyverse)
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## Registered S3 method overwritten by 'rvest':
##   method            from
##   read_xml.response xml2
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

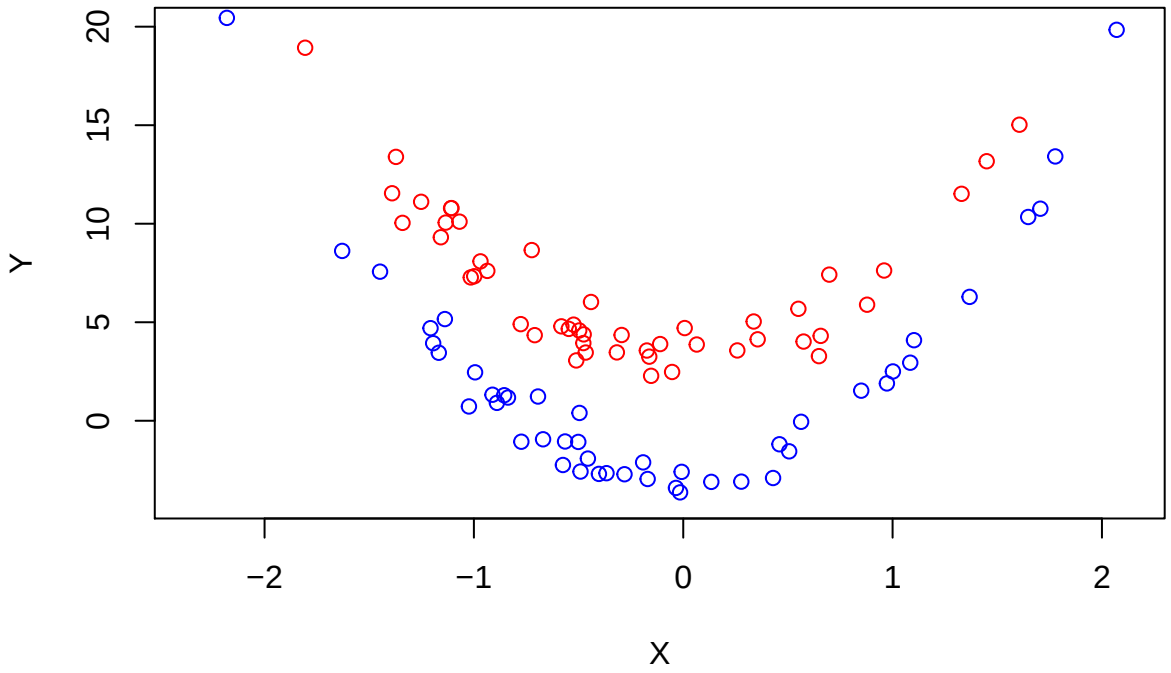
```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
gss_train = read.csv('data/gss_train.csv')
gss_test = read.csv('data/gss_test.csv')
```

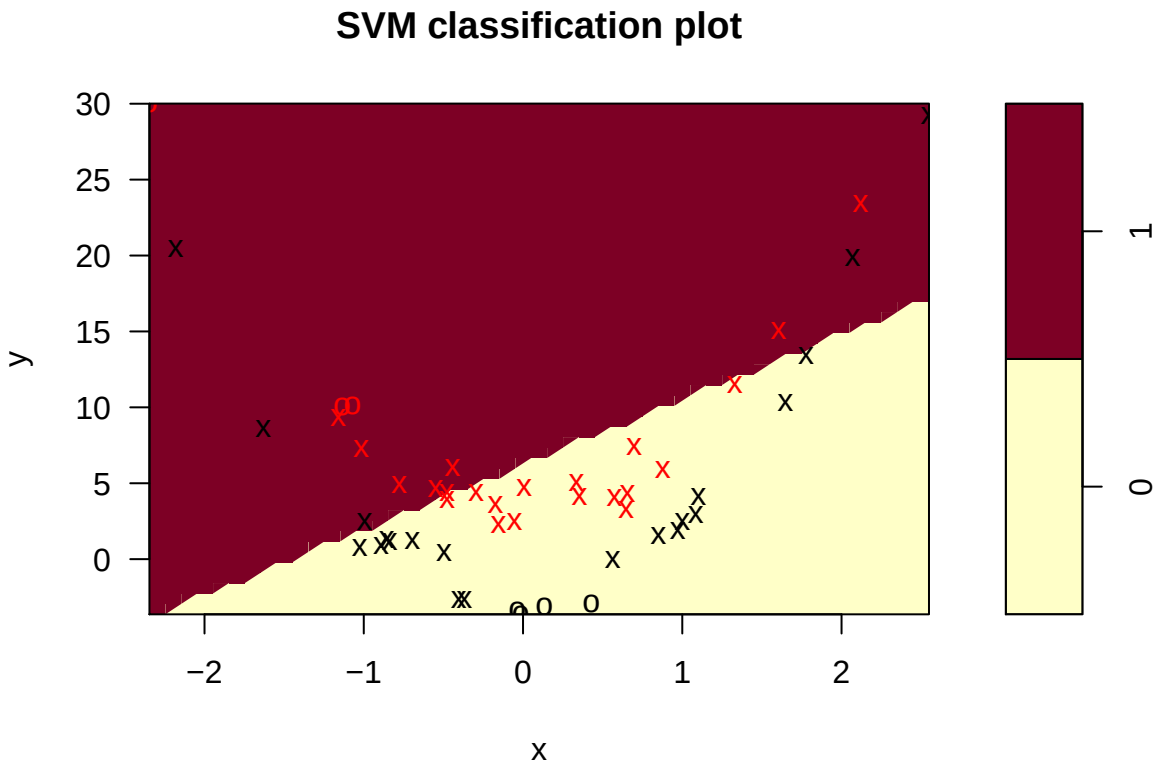
Non-linear separation

```
set.seed(1234)
x <- rnorm(100)
y <- 5*x^2 + rnorm(100)
s <- sample(100, 50)
y[s] <- y[s] + 3
y[-s] <- y[-s] - 3
cl <- rep(0, 100)
cl[s] <- 1
train <- c(sample(s, 25), sample(setdiff(1:100, s), 25))

# Plot
plot(x[s], y[s], col="red", ylim=c(-4, 20), xlab="X", ylab="Y")
points(x[-s], y[-s], col="blue")
```




```
# Train and plot SVM
training <- data.frame(cl = as.factor(cl[train]), y = y[train], x = x[train])
testing <- data.frame(cl = as.factor(cl[-train]), y = y[-train], x = x[-train])
svm.linear <- svm(cl~., data=training, kernel="linear", cost=10)
plot(svm.linear,training)
```




```
# Linear errors
svm_train <- predict(svm.linear, training)
svm_test <- predict(svm.linear, testing)
```

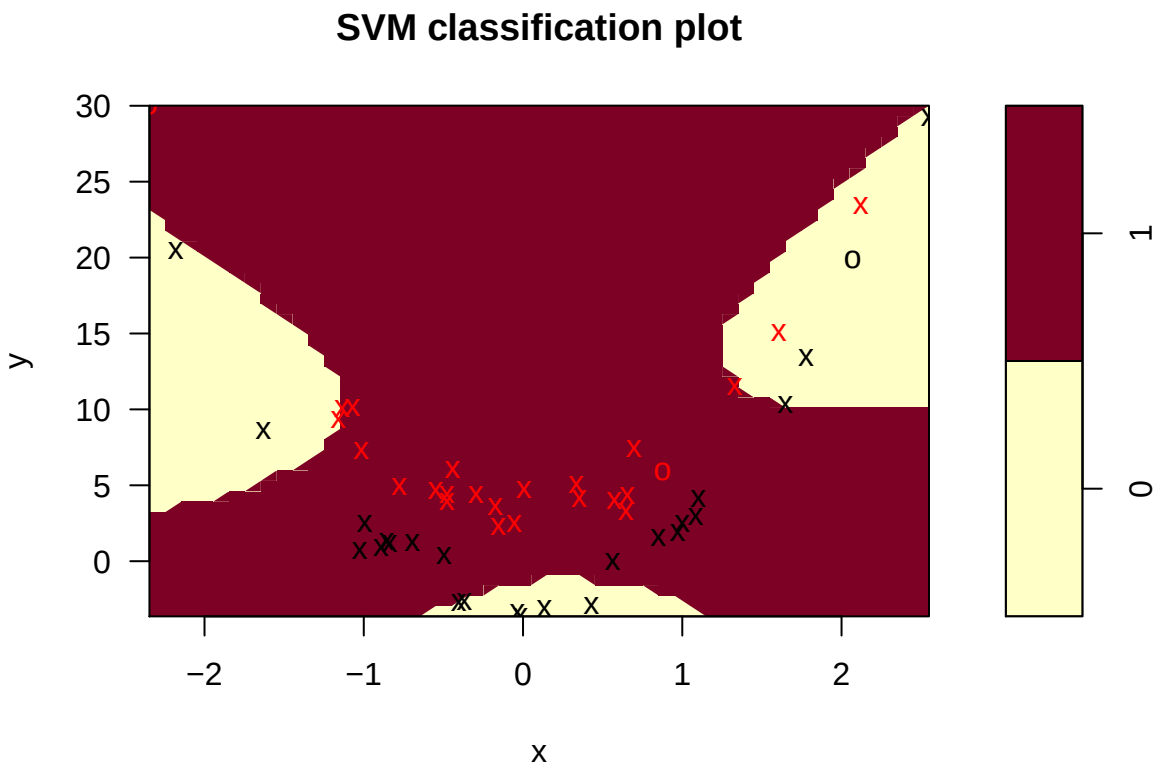
```r
table(cl[train], svm_train)
```

```
##    svm_train
##      0  1
##   0 20  5
##   1 13 12
```

```r
table(cl[-train], svm_test)
```

```
##    svm_test
##      0  1
##   0 19  6
##   1  9 16
```

```r
# Plot polynomial
svm_polynomial <- svm(cl~., data=training, kernel="polynomial", cost=10)
plot(svm_polynomial,training)
```




```r
# Polynomial errors
svm_train <- predict(svm_polynomial, training)
svm_test <- predict(svm_polynomial, testing)
table(cl[train], svm_train)
```

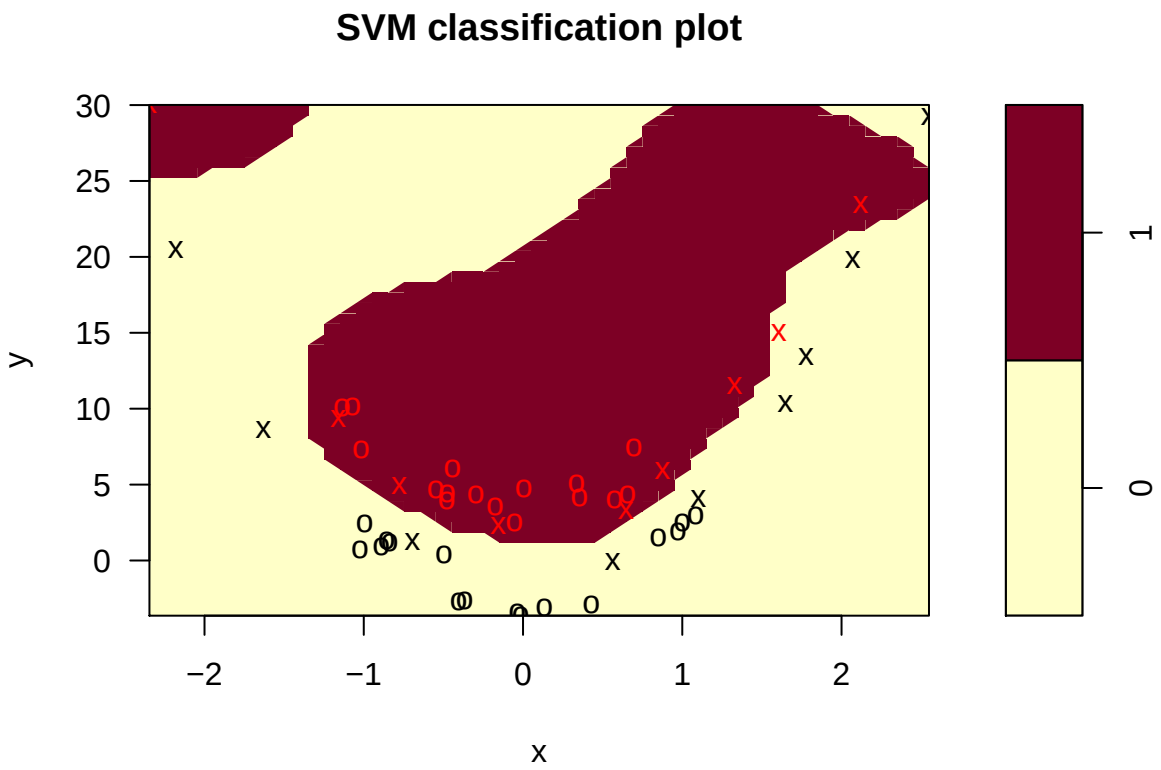
```
##    svm_train
##      0  1
##   0  9 16
##   1  2 23
```

```r
table(cl[-train], svm_test)
```

```
##    svm_test
##      0  1
```

```
##    0  9 16
##    1  5 20
```

```
# Plot radial
svm_radial <- svm(cl~., data=training, kernel="radial", cost=10, gamma=1)
plot(svm_radial,training)
```




```
# Radial errors
svm_train <- predict(svm_radial, training)
svm_test <- predict(svm_radial, testing)
table(cl[train], svm_train)
```

```
##    svm_train
##      0  1
##   0 25  0
##   1  0 25
```

```
table(cl[-train], svm_test)
```

```
##    svm_test
##      0  1
##   0 24  1
##   1  1 24
```

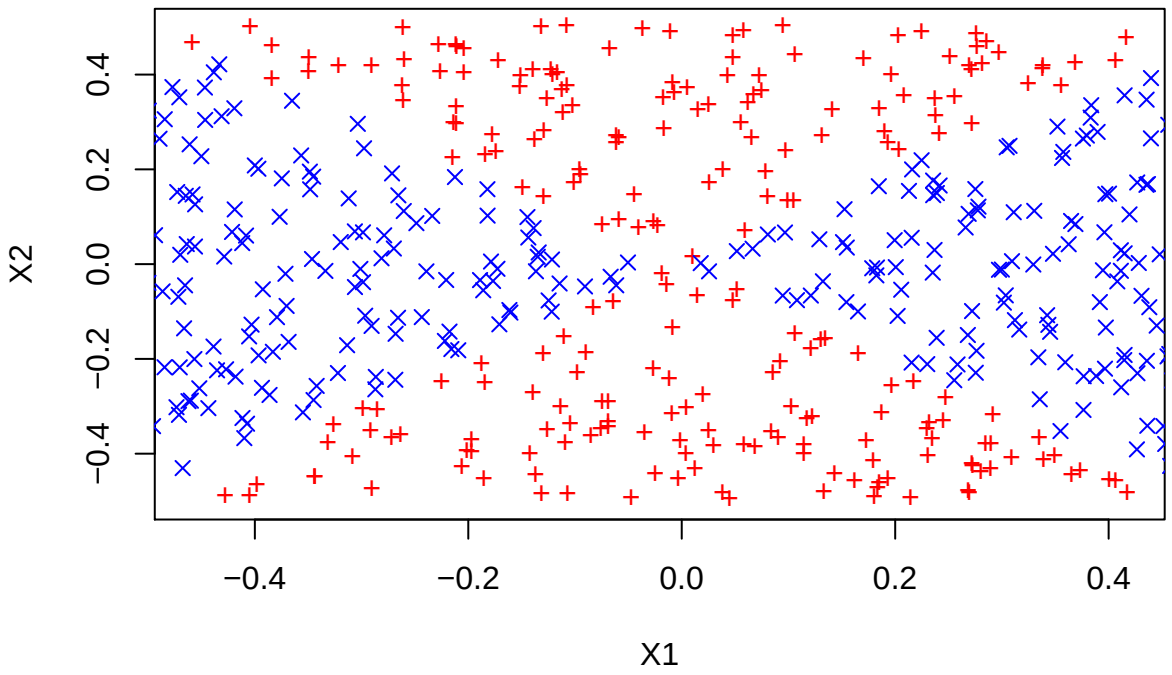
Linear: 4 errors on the training data, 2 errors on the testing data Polynomial: 3 errors on the training data, 1 error on the testing data Radial: 0 errors on the training data, 1 error on the testing data

Radial by far performs the best on the test data in this case, with only error on the testing data

SVM vs. Logistic Regression

```r
# 1.
set.seed(421)
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = 1 * (x1^2 - x2^2 > 0)

# 2.
{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)}
```




```r
# 3.
lm_fit = glm(y ~ x1 + x2, family = binomial)
summary(lm_fit)
```

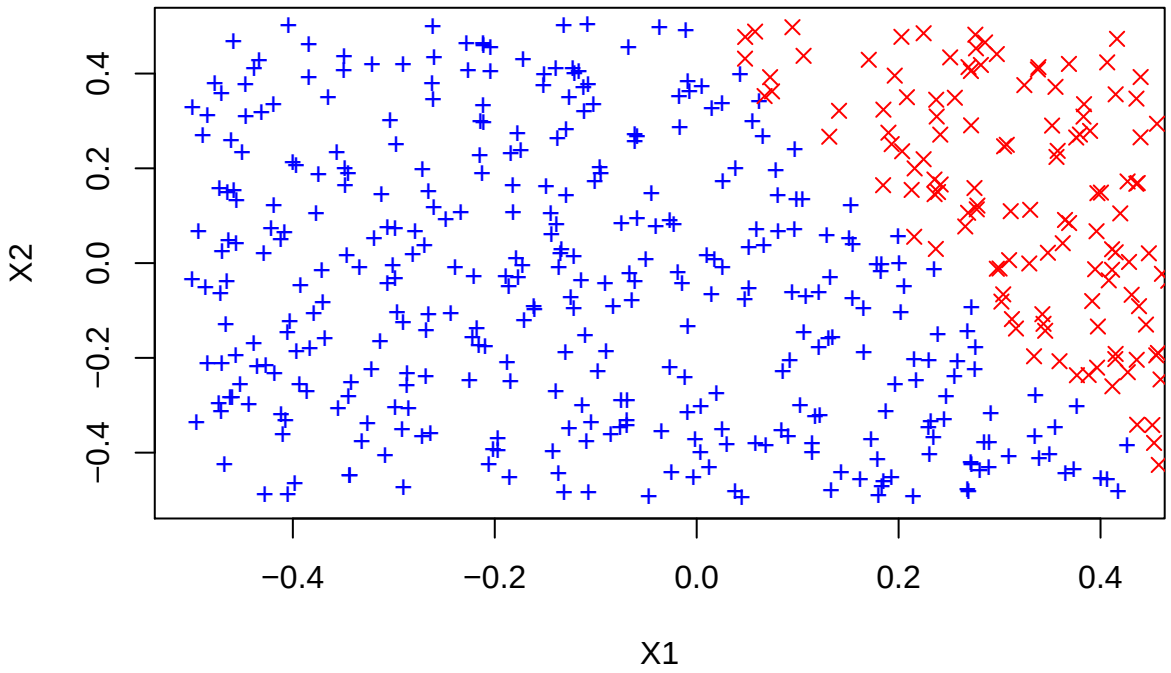
```
##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.278  -1.227   1.089   1.135   1.175
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.11999    0.08971   1.338    0.181
## x1          -0.16881    0.30854  -0.547    0.584
## x2          -0.08198    0.31476  -0.260    0.795
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 691.35  on 499  degrees of freedom
## Residual deviance: 690.99  on 497  degrees of freedom
## AIC: 696.99
```

```
##
## Number of Fisher Scoring iterations: 3
```

```
# 4.
lm_fit = glm(y ~ x1 + x2, family = binomial)
summary(lm_fit)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.278  -1.227   1.089   1.135   1.175
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.11999    0.08971   1.338    0.181
## x1          -0.16881    0.30854  -0.547    0.584
## x2          -0.08198    0.31476  -0.260    0.795
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 691.35  on 499  degrees of freedom
## Residual deviance: 690.99  on 497  degrees of freedom
## AIC: 696.99
##
## Number of Fisher Scoring iterations: 3
```

```
# 5.
data = data.frame(x1 = x1, x2 = x2, y = y)
linear_model = predict(lm_fit, data, type = "response")
linear_prediction = ifelse(linear_model > 0.52, 1, 0)
positive_data = data[linear_prediction == 1, ]
negative_data = data[linear_prediction == 0, ]
{plot(positive_data$x1, positive_data$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(negative_data$x1, negative_data$x2, col = "red", pch = 4)}
```

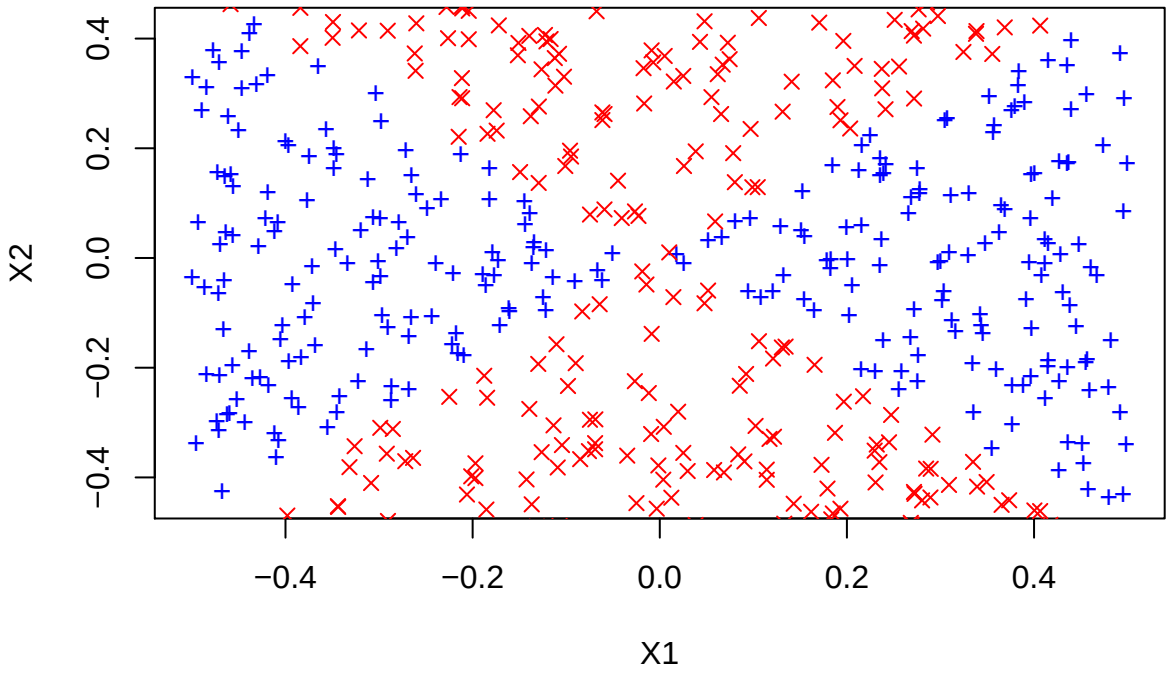



```
# 6.
lm_fit = glm(y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), data = data, family = binomial)
```

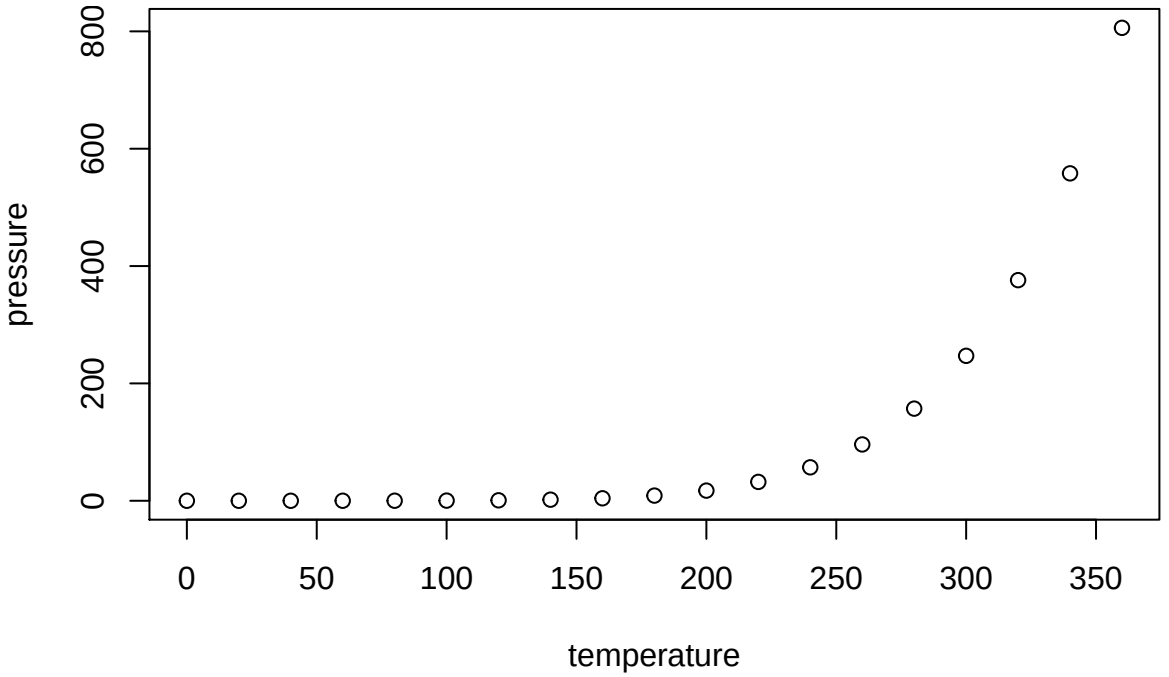
```
## Warning: glm.fit: algorithm did not converge
```

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

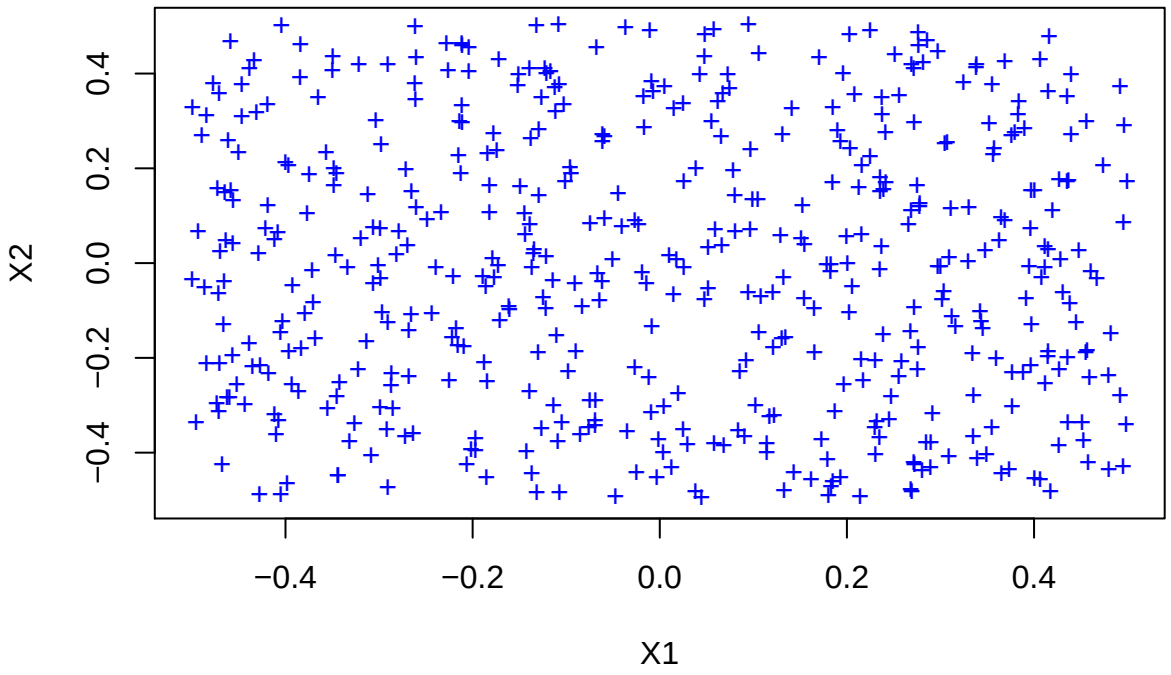
```
# 7.
linear_model = predict(lm_fit, data, type = "response")
linear_prediction = ifelse(linear_model > 0.5, 1, 0)
positive_data = data[linear_prediction == 1, ]
negative_data = data[linear_prediction == 0, ]
{plot(positive_data$x1, positive_data$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(negative_data$x1, negative_data$x2, col = "red", pch = 4)}
```



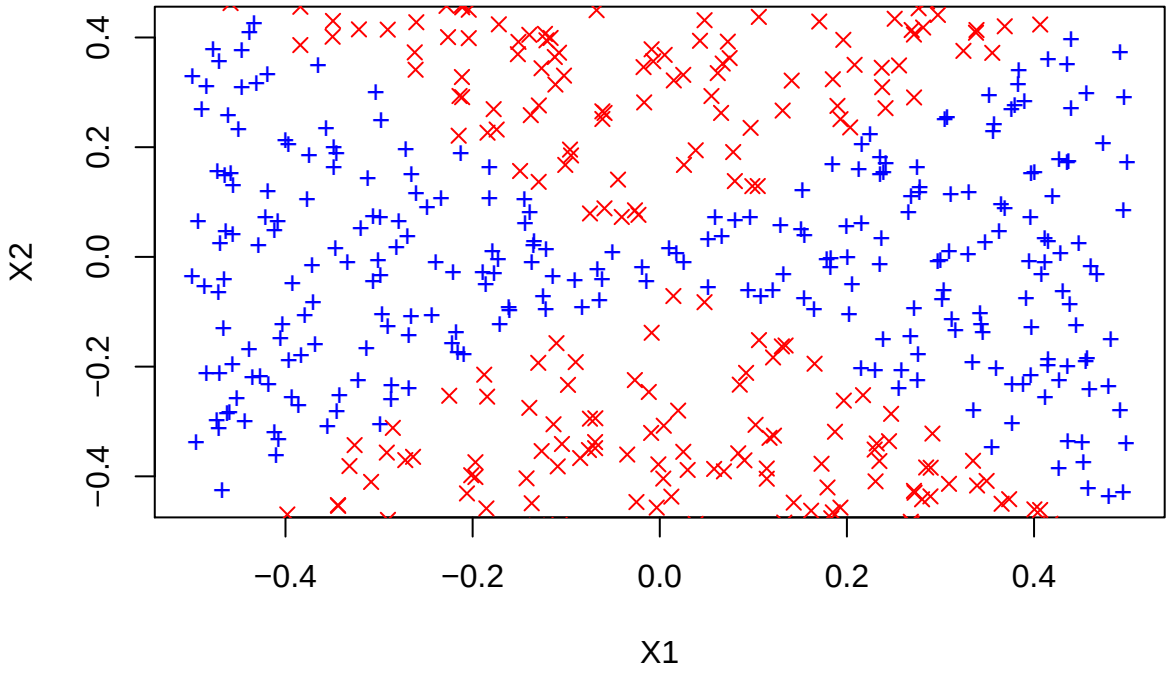

```
plot(pressure)
```




```
# 8.
svm_fit = svm(as.factor(y) ~ x1 + x2, data, kernel = "linear", cost = 0.1)
linear_prediction = predict(svm_fit, data)
positive_data = data[linear_prediction == 1, ]
negative_data = data[linear_prediction == 0, ]
{plot(positive_data$x1, positive_data$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(negative_data$x1, negative_data$x2, col = "red", pch = 4)}
```




```
# 9.
svm_fit = svm(as.factor(y) ~ x1 + x2, kernel = "radial", data, gamma = 1)
radial_prediction = predict(svm_fit, data)
positive_data = data[radial_prediction == 1, ]
negative_data = data[radial_prediction == 0, ]
{plot(positive_data$x1, positive_data$x2, col = "blue", xlab = "X1", ylab = "X2", pch = "+")
points(negative_data$x1, negative_data$x2, col = "red", pch = 4)}
```

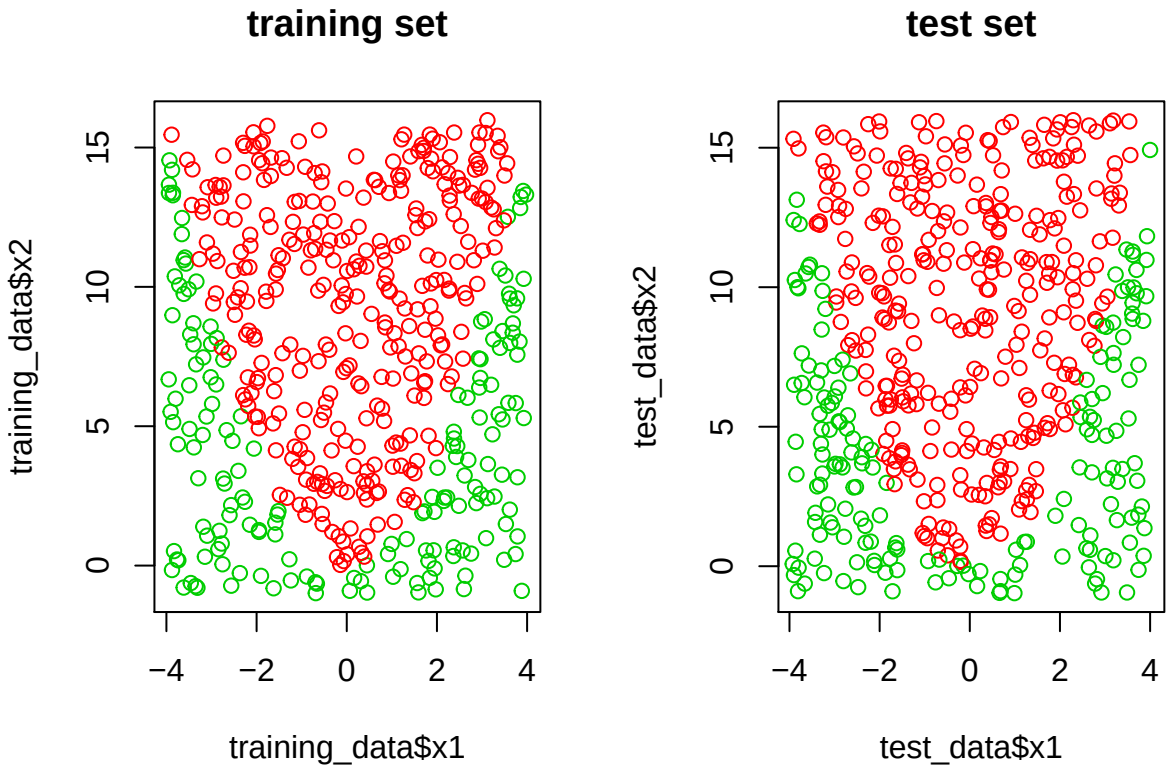


10.

From the above results, it is apparent that both linear logistic regression and linear SVM do a poor job of estimating the above nonlinear decision boundaries. The two models are comparable in performance. Overall, the nonlinear models seem to outperform both of these models.

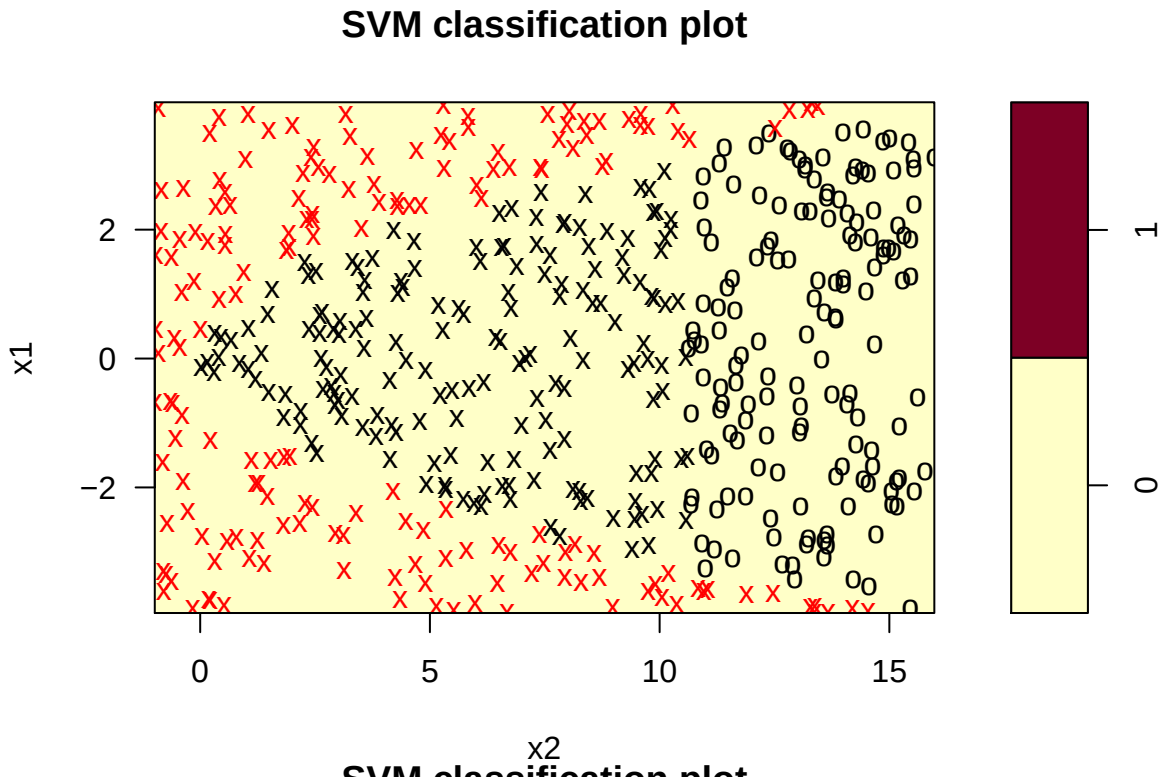
Tuning cost

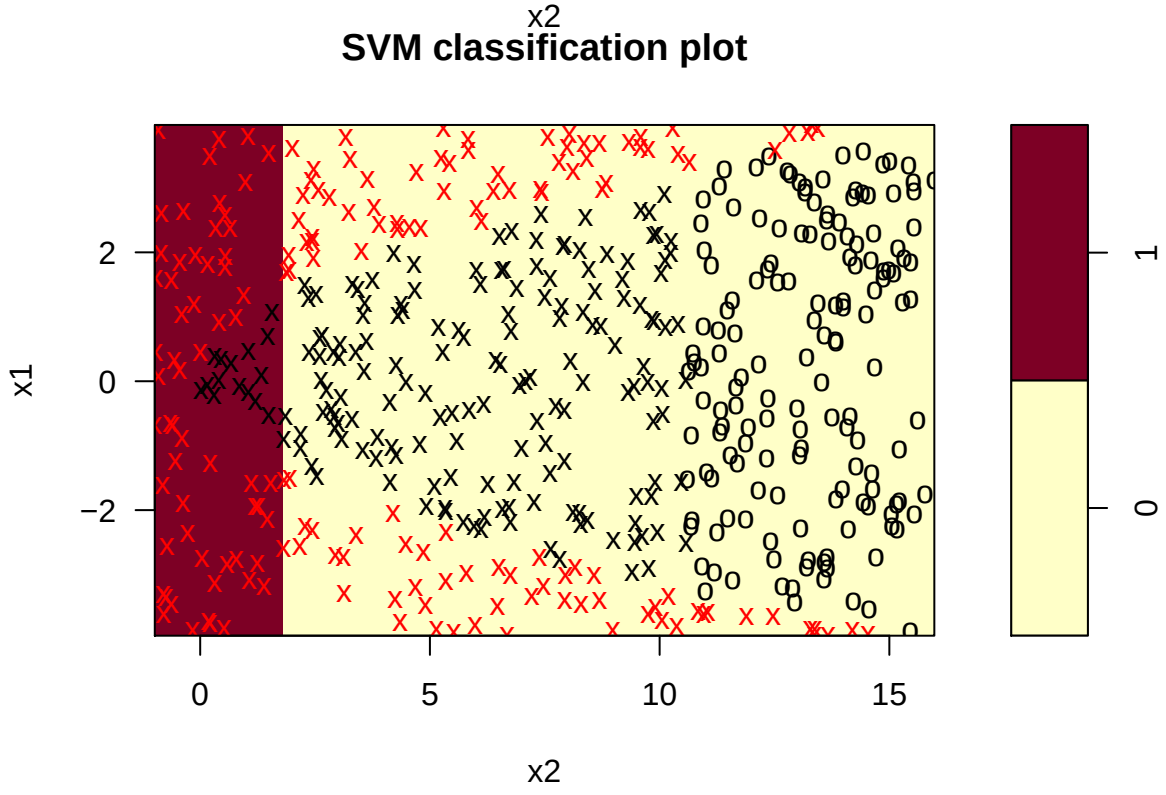
```
# 11.
obs = 1000
x1 <- runif(obs, min = -4, max = 4)
x2 <- runif(obs, min = -1, max = 16)
y <- ifelse(x2 > x1 ^ 2, 0, 1)
dat <- data.frame(x1 = x1, x2 = x2, y = as.factor(y))
train <- sample(obs, obs/2)
training_data <- dat[train, ]
test_data <- dat[-train, ]
par(mfrow = c(1,2))
plot(training_data$x1, training_data$x2, col = as.integer(training_data$y) + 1, main = 'training set')
plot(test_data$x1, test_data$x2, col = as.integer(test_data$y) + 1, main = 'test set')
```

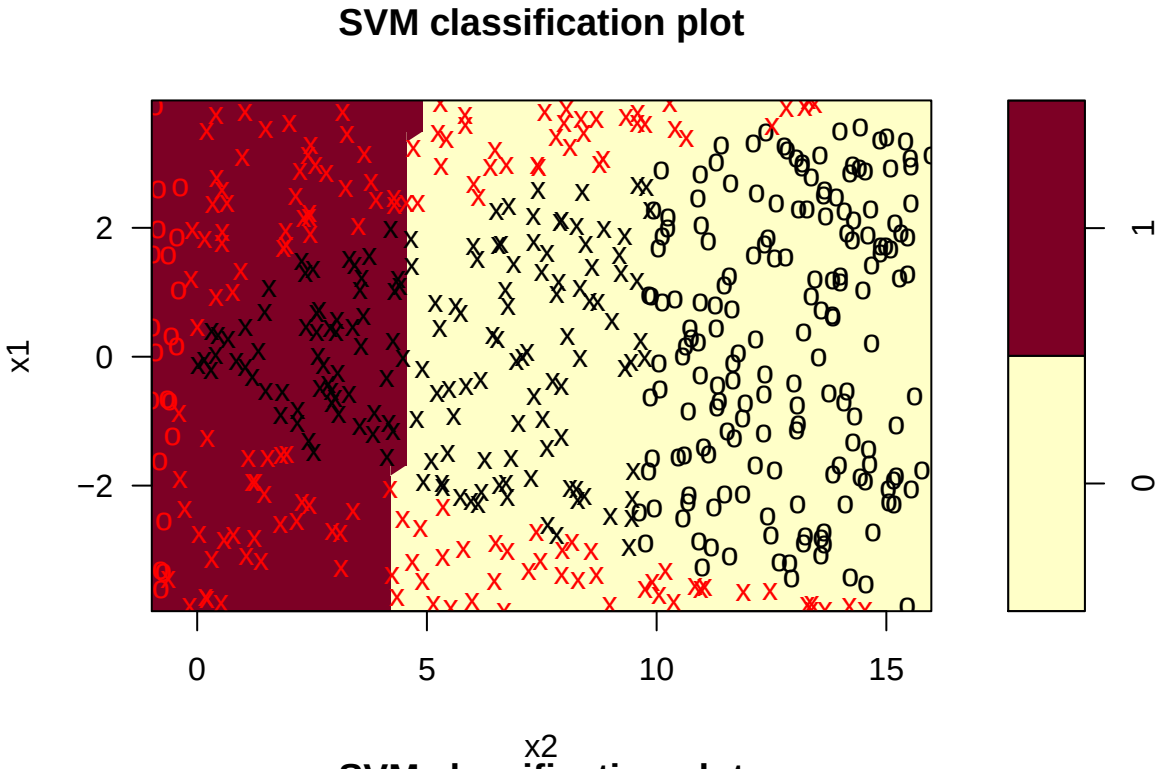


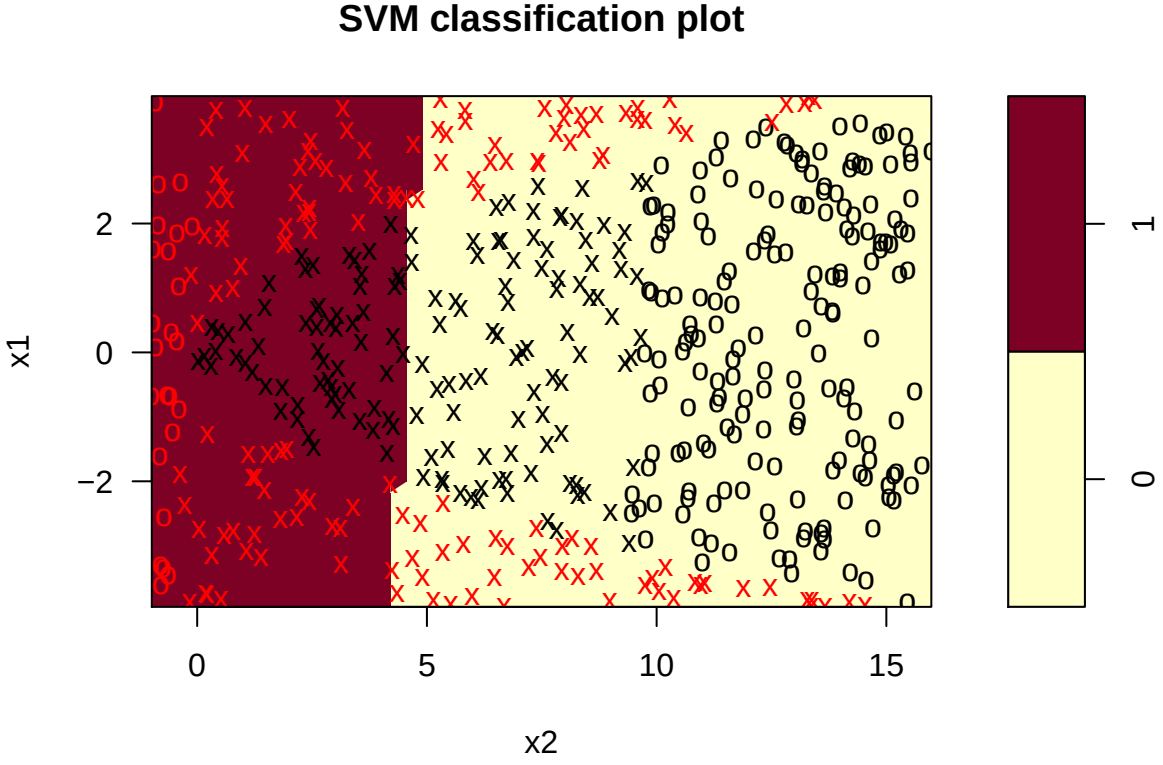

```
# 12.
cost.grid <- c(0.001, 0.01, 0.1, 1, 5, 10, 100)
tune.out <- tune(svm, y ~., data = training_data, kernel = 'linear', ranges = list(cost = cost.grid))
err.rate.train <- rep(NA, length(cost.grid))
for (cost in cost.grid) {
  svm.fit <- svm(y ~ ., data = training_data, kernel = 'linear', cost = cost)
  plot(svm.fit, data = training_data)
  res <- table(prediction = predict(svm.fit, newdata = training_data), truth = training_data$y)
  err.rate.train[match(cost, cost.grid)] <- (res[2,1] + res[1,2]) / sum(res)
```

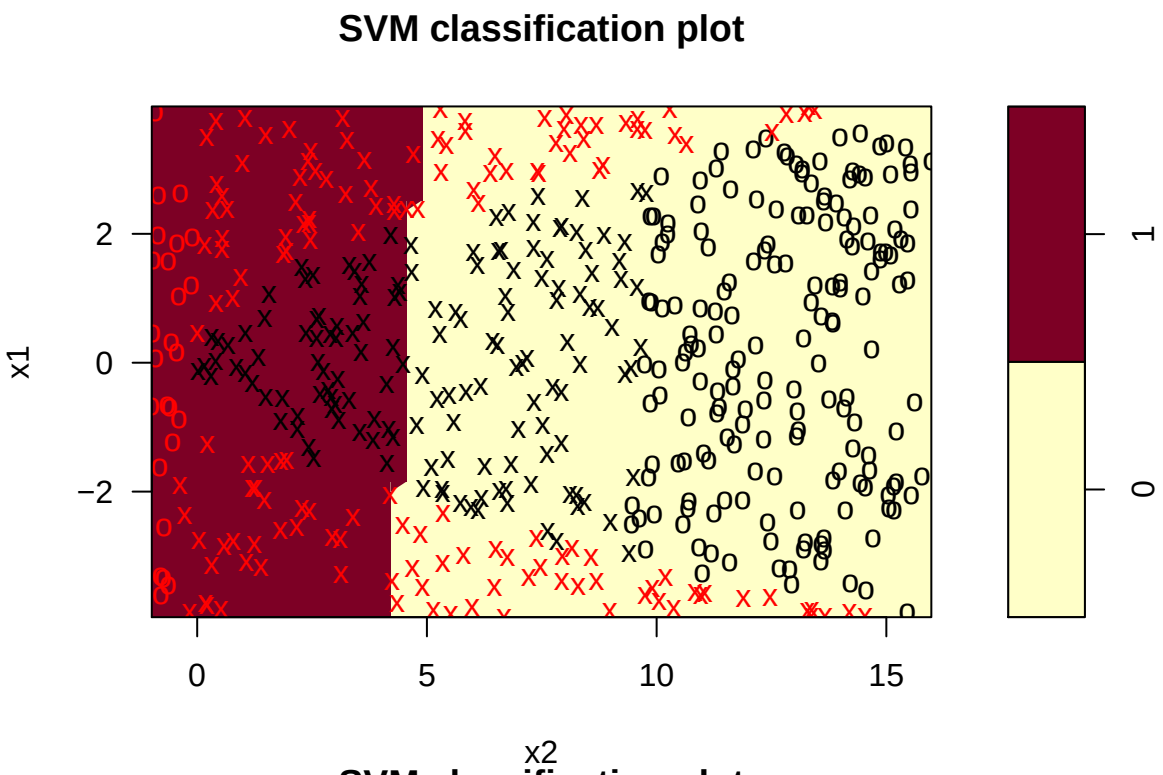
```
}
```

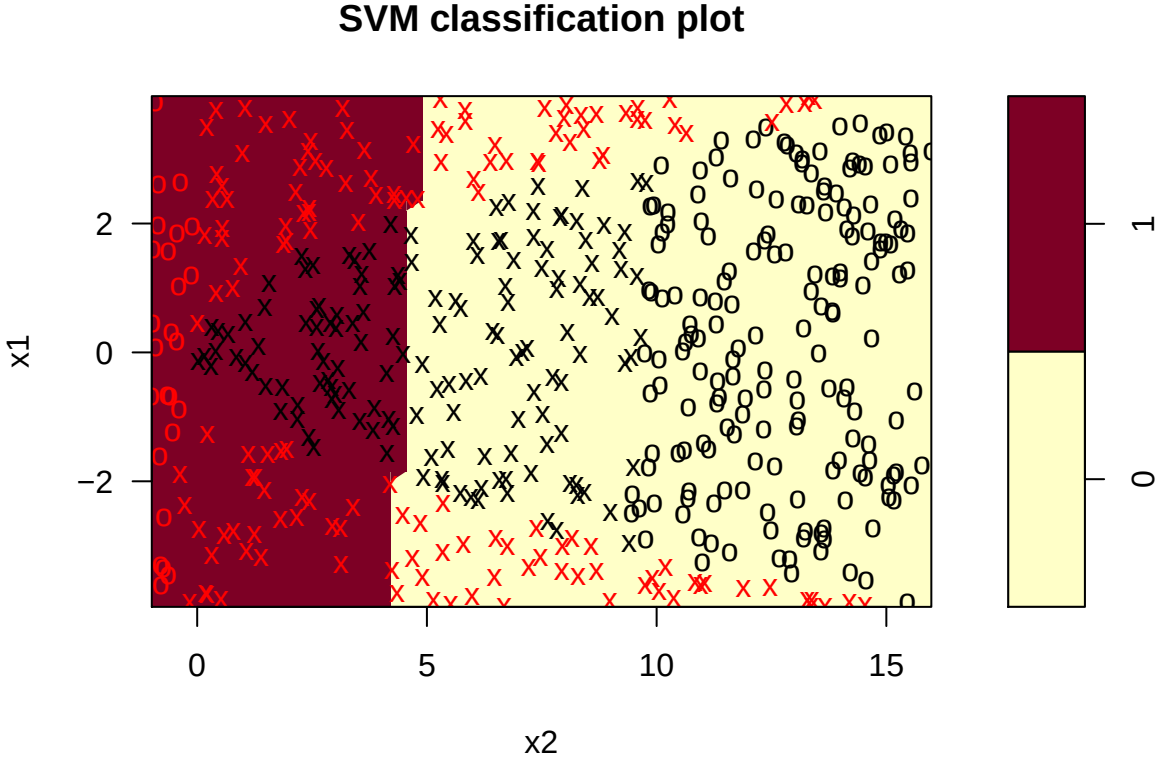

SVM classification plot
x1
x2
2
0
−2
0
5
10
15
1
0


SVM classification plot
x1
x2
2
0
−2
0
5
10
15
1
0


SVM classification plot
x1
x2
2
0
−2
0
5
10
15
1
0


SVM classification plot
x1
x2
2
0
−2
0
5
10
15
1
0





The cost, 0.01, has the minimum training error: 0.254

As the cost increases, the training errors appear to decrease until reaching 0.1, after which point they begin to increase again. The minimum training error is obtained for cost = 0.1. This may be because of the nonlinear separation boundary where a linear SVM is prone to overfitting at high levels of cost.

```r
# 13.
err.rate.test <- rep(NA, length(cost.grid))
for (cost in cost.grid) {
  svm.fit <- svm(y ~ ., data = training_data, kernel = 'linear', cost = cost)
  res <- table(prediction = predict(svm.fit, newdata = test_data), truth = test_data$y)
  err.rate.test[match(cost, cost.grid)] <- (res[2,1] + res[1,2]) / sum(res)
}
err.rate.test
```

```
## [1] 0.352 0.252 0.272 0.272 0.272 0.272 0.272
```

The cost, 0.01, has the minimum training error: 0.254

The optimal result is consistent with the cross-validation result, and both are optimal when cost = 0.1.

14.

Results seem to indicate that higher cost reduces error rates, but after a point, this reduction reverses itself (cost rising from 10 to 100). The two classes as observed in the first scatter diagram do not have a clean linear separation boundary, and therefore, high cost makes the model sensitive to overfitting concerns.

```r
library(caret)
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(kernlab)
```

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

```
## The following object is masked from 'package:purrr':
##
##     cross
```

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

```r
library(knitr)

# 15
costs <- c(0.001, 0.01, 0.1, 1, 5, 10)

svm_linear <- train(
  y = as.factor(gss_train$colrac),
  x = scale(gss_train %>% select(- colrac)),
  method = 'svmLinear',
  trControl = trainControl(method = 'cv',
                           number = 10,
                           savePredictions = 'final'),
  tuneGrid = expand.grid(C = costs)
)
```

```r
svm_linear$results %>%
  mutate(cv_error = 1 - Accuracy) %>%
  select(C, cv_error) %>%
  arrange(cv_error)
```

```
##       C  cv_error
## 1 1e-02 0.2018945
## 2 1e-01 0.2072728
## 3 1e+00 0.2072774
## 4 5e+00 0.2086196
## 5 1e+01 0.2086196
## 6 1e-03 0.2208231
```

```r
# 16

gammas <- c(0.001, 0.01, 0.1, 1)

svm_polynomial <- train(
  y = as.factor(gss_train$colrac),
  x = scale(gss_train %>% select(- colrac)),
  method = 'svmPoly',
  trControl = trainControl(method = 'cv',
                           number = 10,
                           savePredictions = 'final'),
  tuneGrid = expand.grid(C = costs, degree = c(2:5), scale = 1)
)

svm_radial <- train(
  y = as.factor(gss_train$colrac),
  x = scale(gss_train %>% select(- colrac)),
  method = 'svmRadial',
  trControl = trainControl(method = 'cv',
                           number = 10,
                           savePredictions = 'final'),
  tuneGrid = expand.grid(C = costs, sigma = gammas)
)

svm_polynomial$results %>%
  mutate(cv_error = 1 - Accuracy) %>%
  select(C, degree, cv_error) %>%
  arrange(cv_error)
```

```
##        C degree  cv_error
## 1  1e-03      5 0.2121080
## 2  1e-02      5 0.2121080
## 3  1e-01      5 0.2121080
## 4  1e+00      5 0.2121080
## 5  5e+00      5 0.2121080
## 6  1e+01      5 0.2121080
## 7  1e-03      3 0.2168287
## 8  1e-02      3 0.2168287
## 9  1e-01      3 0.2168287
## 10 1e+00      3 0.2168287
## 11 5e+00      3 0.2168287
```

```
## 12 1e+01      3 0.2168287
## 13 1e-03      4 0.2424273
## 14 1e-02      4 0.2424273
## 15 1e-01      4 0.2424273
## 16 1e+00      4 0.2424273
## 17 5e+00      4 0.2424273
## 18 1e+01      4 0.2424273
## 19 1e-03      2 0.2451483
## 20 1e-02      2 0.2795948
## 21 1e-01      2 0.2870274
## 22 1e+00      2 0.2870274
## 23 5e+00      2 0.2870274
## 24 1e+01      2 0.2870274
```

```
svm_radial$results %>%
  mutate(cv_error = 1 - Accuracy) %>%
  select(C, sigma, cv_error) %>%
  arrange(cv_error)
```

```
##        C sigma  cv_error
## 1  1e+00 0.010 0.1951103
## 2  1e+01 0.001 0.2032460
## 3  5e+00 0.001 0.2072774
## 4  5e+00 0.010 0.2086470
## 5  1e+00 0.001 0.2106467
## 6  1e+01 0.010 0.2127421
## 7  1e-01 0.010 0.2180792
## 8  1e-01 0.001 0.2573151
## 9  5e+00 0.100 0.2976921
## 10 1e+01 0.100 0.2976921
## 11 1e+00 0.100 0.3146254
## 12 1e-03 0.001 0.4746776
## 13 1e-02 0.001 0.4746776
## 14 1e-03 0.010 0.4746776
## 15 1e-02 0.010 0.4746776
## 16 1e-03 0.100 0.4746776
## 17 1e-02 0.100 0.4746776
## 18 1e-01 0.100 0.4746776
## 19 1e-03 1.000 0.4746776
## 20 1e-02 1.000 0.4746776
## 21 1e-01 1.000 0.4746776
## 22 1e+00 1.000 0.4746776
## 23 5e+00 1.000 0.4746776
## 24 1e+01 1.000 0.4746776
```

```
# Test set comparison

test_colrac = scale(gss_train %>% select(- colrac))
test_linear = mean(predict(svm_linear, test_colrac != gss_test$colrac))
```

```
## Warning in test_colrac != gss_test$colrac: longer object length is not a
## multiple of shorter object length
```

```
## Warning in mean.default(predict(svm_linear, test_colrac !=
## gss_test$colrac)): argument is not numeric or logical: returning NA
```

```
test_polynomial = mean(predict(svm_polynomial, test_colrac != gss_test$colrac))
```

```
## Warning in test_colrac != gss_test$colrac: longer object length is not a
## multiple of shorter object length
```

```
## Warning in mean.default(predict(svm_polynomial, test_colrac !=
## gss_test$colrac)): argument is not numeric or logical: returning NA
```

```
test_radial = mean(predict(svm_radial, test_colrac != gss_test$colrac))
```

```
## Warning in test_colrac != gss_test$colrac: longer object length is not a
## multiple of shorter object length
```

```
## Warning in mean.default(predict(svm_radial, test_colrac !=
## gss_test$colrac)): argument is not numeric or logical: returning NA
```

For the linear SVM, 10-fold CV reveals that the min error rate is 0.2012. The polynomial SVM generally produces the highest error rates, where a degree 3 polynomial performs best across all cost factors, but doesn't exhibit a lot of variation in error rates across all hyperparameters.

The lowest error radial CV (Cost = 1, Gamma = 0.01) outperforms both linear SVM and polynomial SVM, though the variation in radial CV rates is quite high across the various combinations of hyperparameters.