# Support Vector Machines Exercises Ch9

*Darshan Patel*

*3/1/2019*

The following set of problems are from the applied exercises section in ISLR Chapter 9: Support Vector Machines.

```
rm(list = ls())
library(MASS)
library(ISLR)
library(tidyverse)
```

```
## Warning: package 'tibble' was built under R version 3.4.4
```

```
## Warning: package 'tidyr' was built under R version 3.4.4
```

```
## Warning: package 'purrr' was built under R version 3.4.4
```

```
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
library(gridExtra)
library(e1071)
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.4.4
```

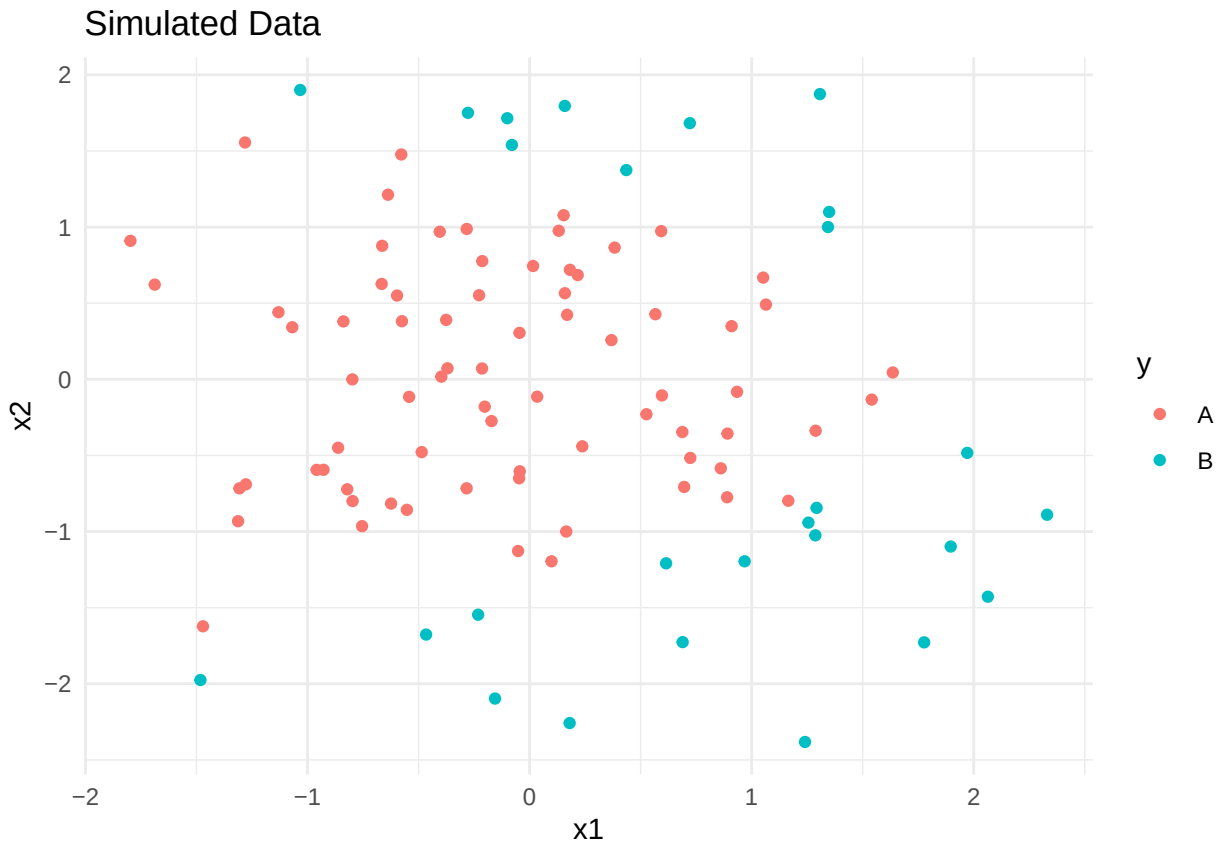
```
## Warning: package 'Matrix' was built under R version 3.4.4
```

**Question 4: Generate a simulated two-class 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 assertions.**

```
set.seed(4)
x1 = rnorm(100)
x2 = rnorm(100)
df = data.frame(x1, x2,
                y = as.factor(ifelse(x1 + x2^2 < 2,
                                     "A", "B")))
```

This is a visualization of the data where there is a visible but non-linear separation between the two classes.

```
ggplot(df, aes(x = x1, y = x2, color = y)) + geom_point() +
  ggtitle("Simulated Data") +
  theme_minimal()
```




First create a train/test split.

```
set.seed(12)
indices = sample(1:nrow(df), size = 0.7*nrow(df))
train = df[indices,]
test = df[-indices,]
```

Then create a support vector machine with a polynomial kernel and a support vector machine with a radial kernel.

```
svm_model1 = svm(y~., data = train,
                 kernel = "polynomial", degree = 2, cost = 1)
svm_model2 = svm(y~., data = train,
                 kernel = "radial", gamma = 1, cost = 1)
```

Compare the two on the train and test data.

```
summary(svm_model1)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "polynomial", degree = 2,
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
```

```
##      degree:  2
##       gamma:  0.5
##      coef.0:  0
##
## Number of Support Vectors:  23
##
##  ( 12 11 )
##
##
## Number of Classes:  2
##
## Levels:
##  A B
```

```
summary(svm_model2)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "radial", gamma = 1,
##     cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  30
##
##  ( 15 15 )
##
##
## Number of Classes:  2
##
## Levels:
##  A B
```

The SVM with the radial kernel has 30 support vectors while the SVM with the polynomial kernel has 23 support vectors.

The training error rates of the two models, polynomial and radial kernel respectively, are:

```
(nrow(train) - sum(diag(table(predict(svm_model1),
                              train$y)))) / nrow(train)
```

```
## [1] 0.1142857
```

```
(nrow(train) - sum(diag(table(predict(svm_model2),
                              train$y)))) / nrow(train)
```

```
## [1] 0.02857143
```

The radial kernel SVM created less error on the training data.

Below is the confusion matrix for both models, polynomial kernel and radial kernel respectively, on the test data.

```
table(predict(svm_model1, test, type = "response"), test$y)
```

```
##
##       A  B
##    A 19  2
##    B  3  6
```

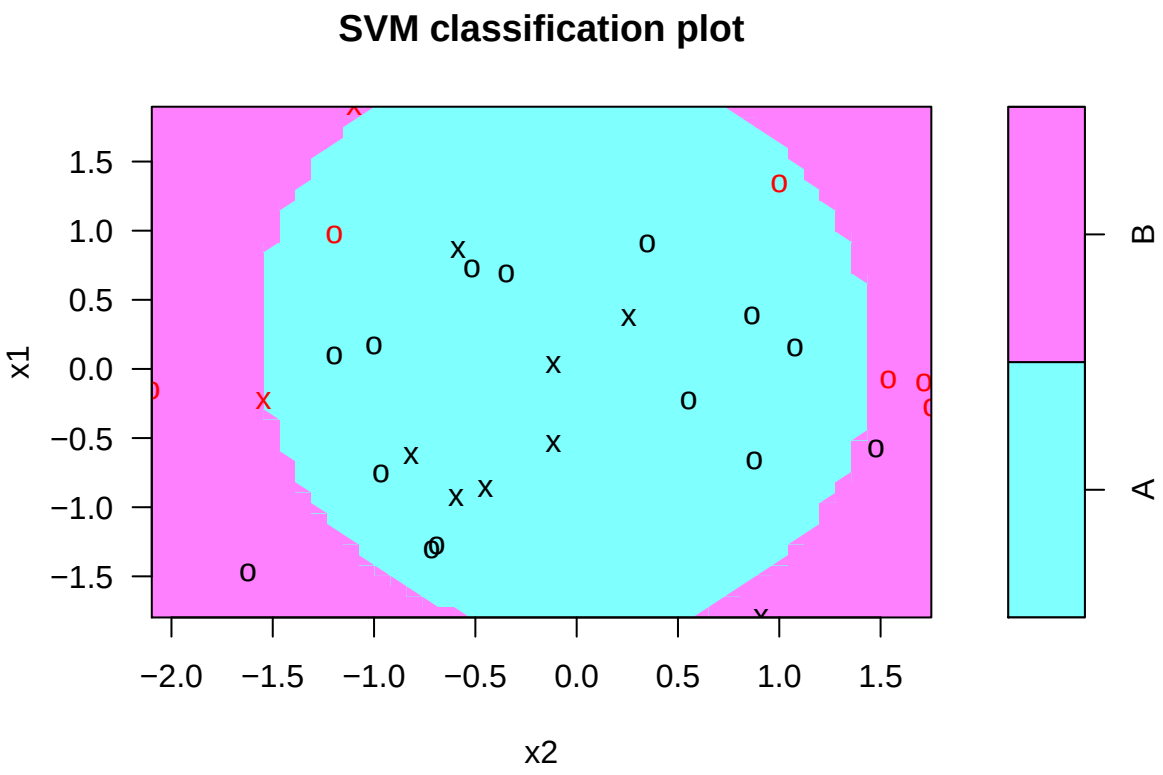
```
table(predict(svm_model2, test, type = "response"), test$y)
```

```
##
##       A  B
##    A 21  0
##    B  1  8
```

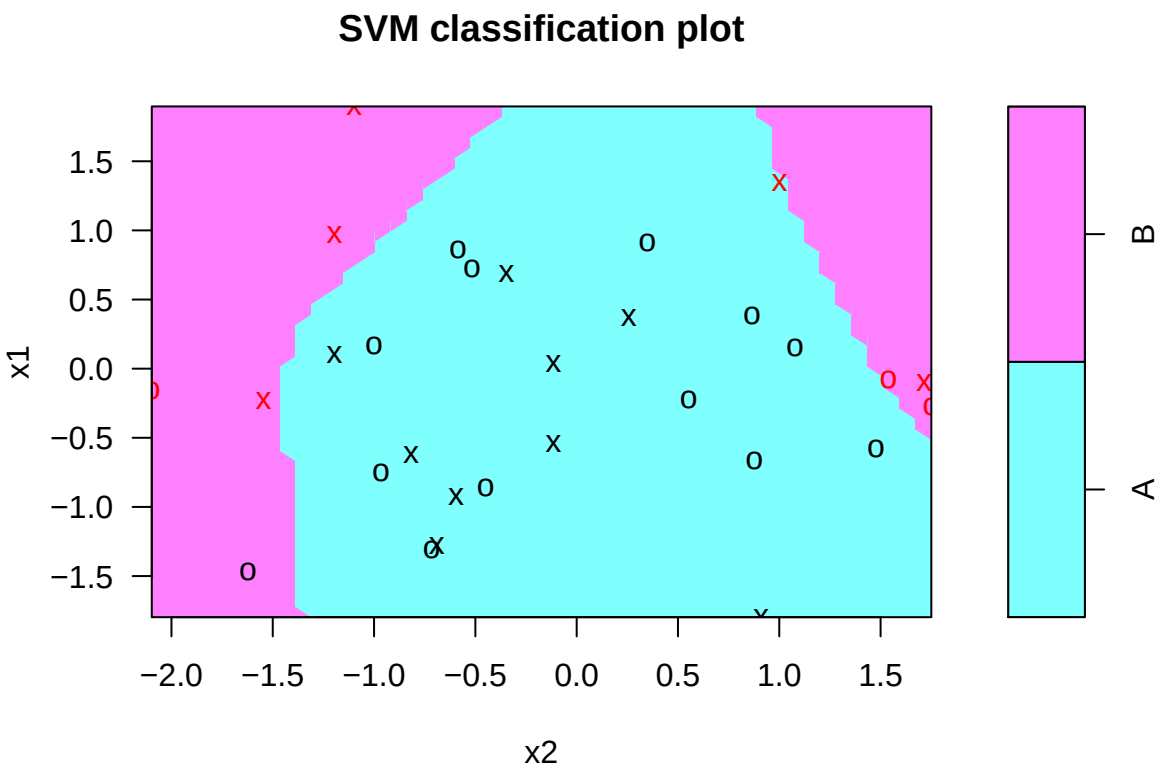
However, the SVM with the radial kernel performed better in classification on the test data than the SVM with the polynomial kernel.

A plot of the SVM classification on the test data is shown below.

```
plot(svm_model1, test)
```




```
plot(svm_model2, test)
```




It is clearly shown that in the second (radial kernel) plot, there is only one error made whereas in the first (polynomial kernel) plot, there are more errors made.

The test error rates are:

```
(nrow(test) - sum(diag(table(predict(svm_model1, test),
                             test$y)))) / nrow(test)
```

```
## [1] 0.1666667
```

```
(nrow(test) - sum(diag(table(predict(svm_model2, test),
                             test$y)))) / nrow(test)
```

```
## [1] 0.03333333
```

The test error rates are greater than their training error rates, respectively. However, the radial kernel model continues to be the better model for classification on this dataset.

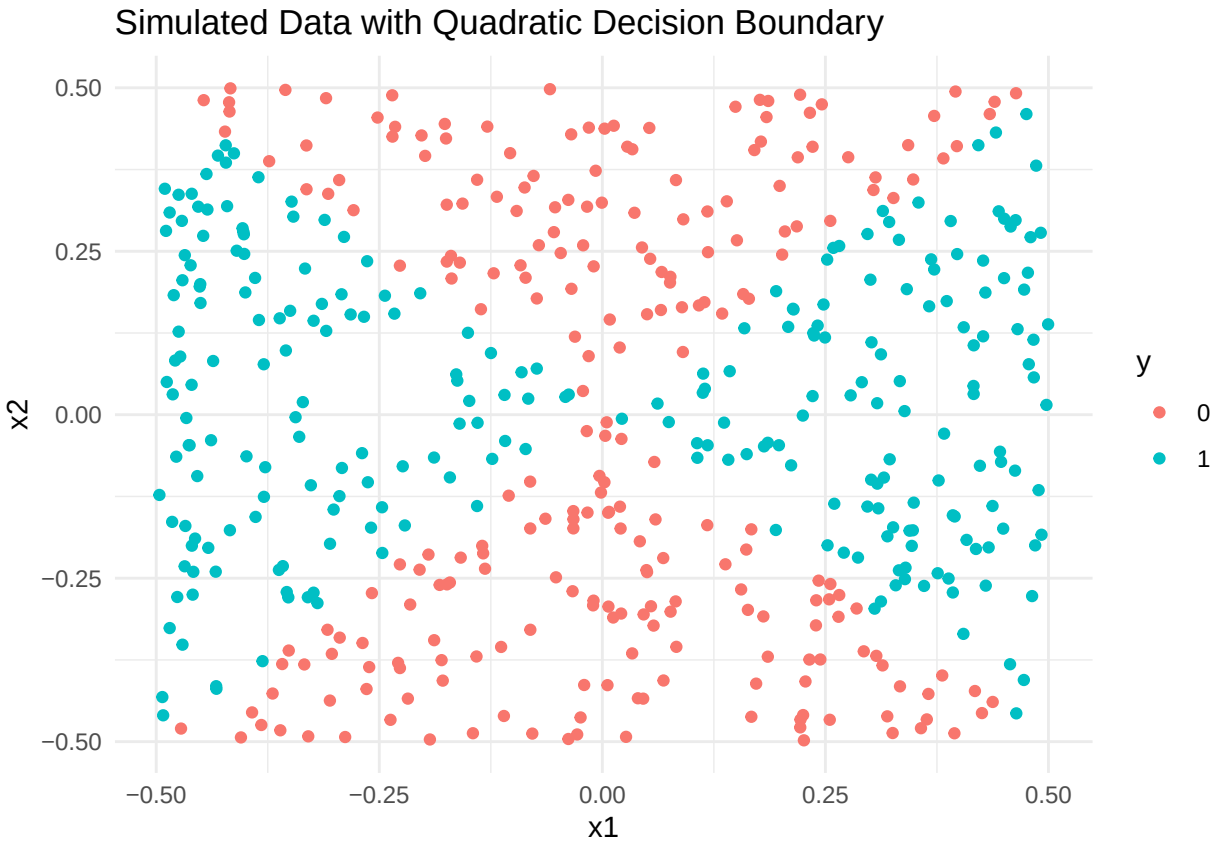
**Question 5: We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.**

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

```
set.seed(500)
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = 1*(x1^2 - x2^2 > 0)
df = data.frame(x1, x2, y = as.factor(y))
```

(b) Plot the observations, colored according to their class labels. The plot should display $X_1$ on the $x$-axis and $X_2$ on the $y$-axis.

```
ggplot(df, aes(x = x1, y = x2, color = y)) + geom_point() +
  ggtitle("Simulated Data with Quadratic Decision Boundary") +
  theme_minimal()
```

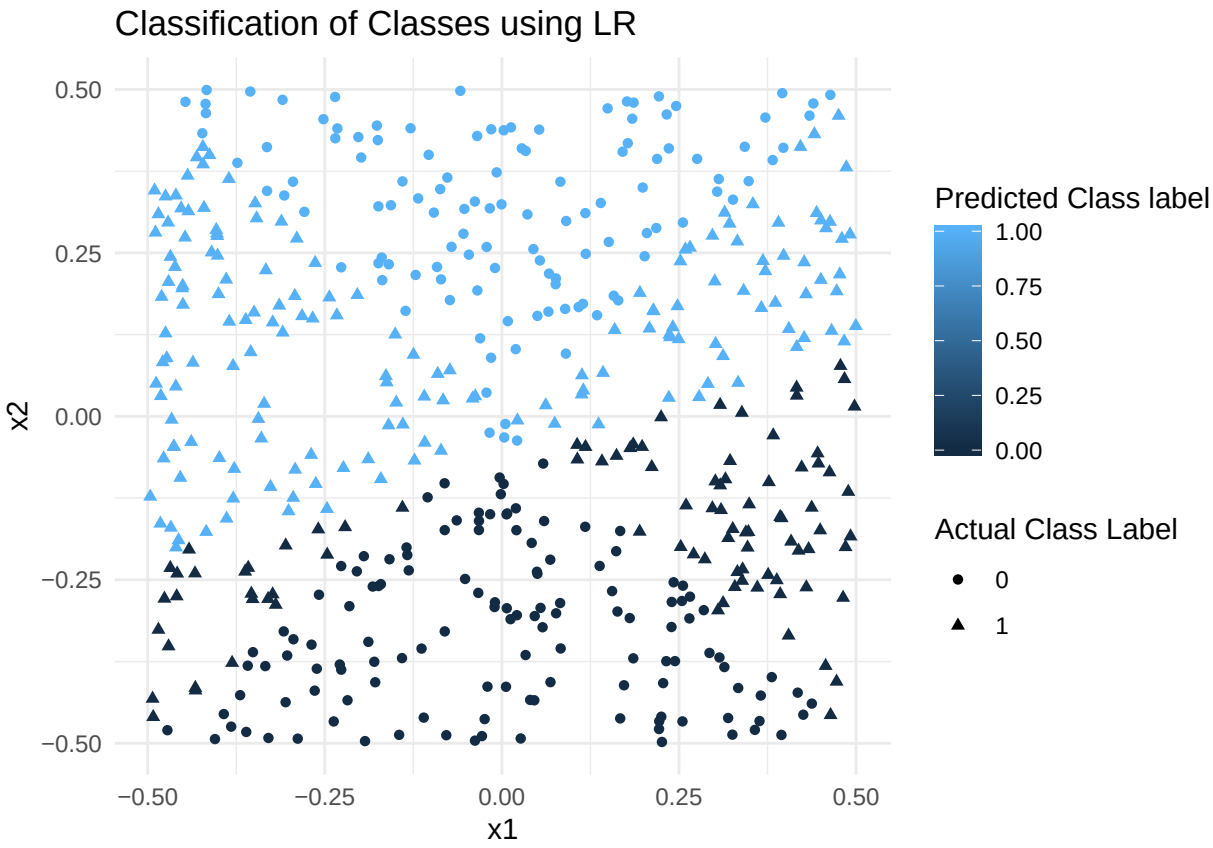



(c) Fit a logistic regression model to the data, using $X_1$ and $X_2$ as predictors.

```
model3 = glm(y~., data = df, family = "binomial")
```

(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.

```
ggplot(df, aes(x = x1, y = x2,
               color = ifelse(predict(model3) < 0,
                              0, 1),
               pch = y)) +
  geom_point() +
  labs(color = "Predicted Class label",
       pch = "Actual Class Label") +
  ggtitle("Classification of Classes using LR") +
  theme_minimal()
```




The decision boundary between the two classes is clearly linear.

(e) Now fit a logistic regression model to the data using non-linear functions of $X_1$ and $X_2$ as predictors (e.g. $X_1^2$, $X_1 \times X_2$, $\log(X_2)$, and so forth).

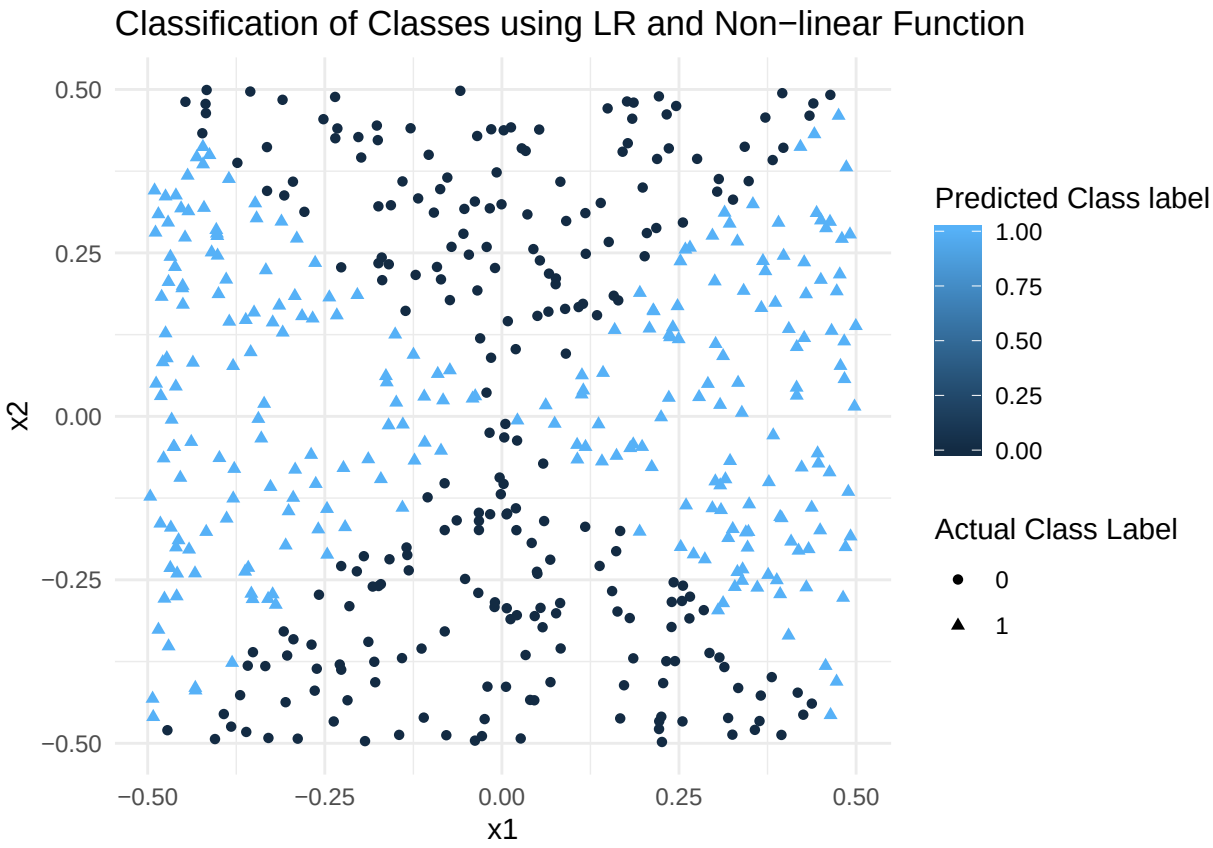
```
model4 = glm(y~poly(x1, 2) + poly(x2, 2), data = df, family = "binomial")
```

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

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

(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 an example in which the predicted class labels are obviously non-linear.

```
ggplot(df, aes(x = x1, y = x2, color = ifelse(predict(model4) < 0,
                                              0, 1),
               pch = y)) +
  geom_point() +
  labs(color = "Predicted Class label",
       pch = "Actual Class Label") +
  ggtitle("Classification of Classes using LR and Non-linear Function") +
  theme_minimal()
```

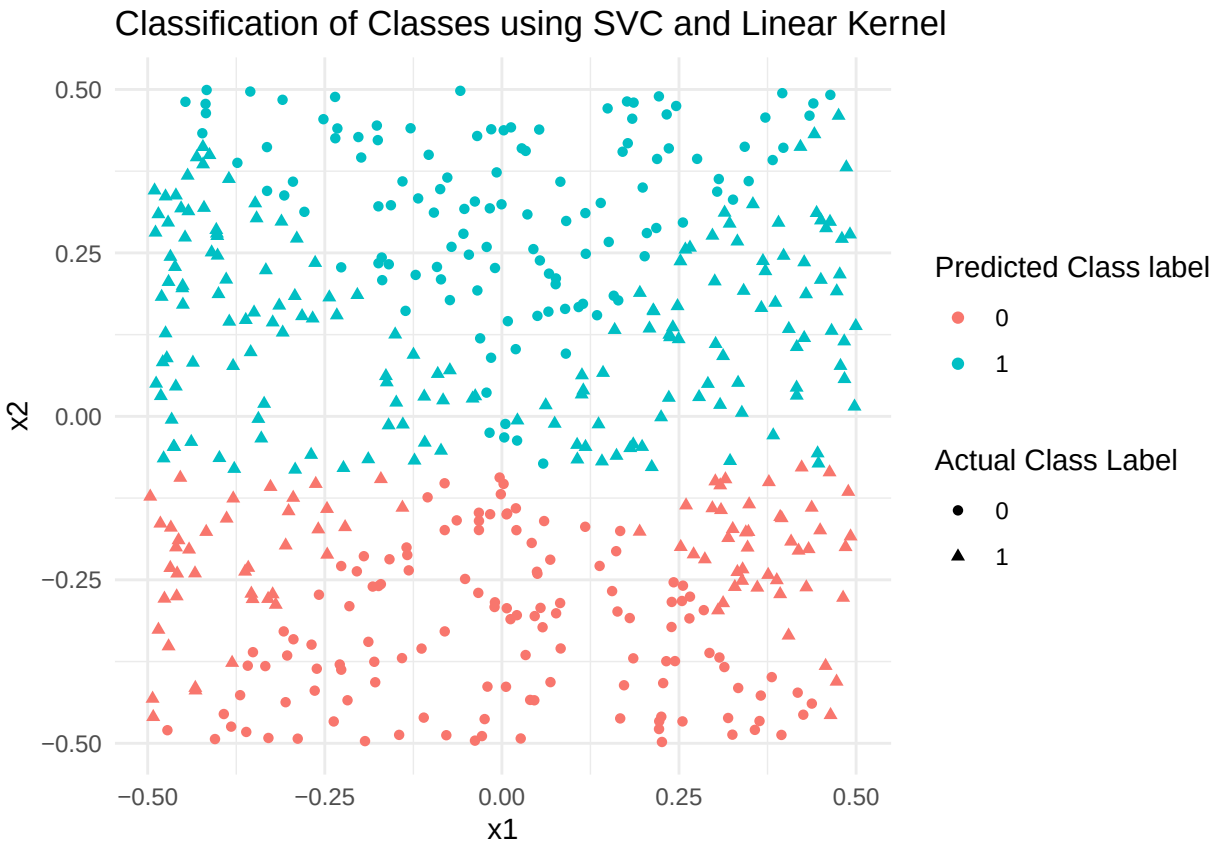



The decision boundary here is not linear.

(g) Fit a support vector classifier to the data with $X_1$ and $X_2$ as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the *predicted class labels*.

```
model5 = svm(y~., df, kernel = "linear")
predictions = predict(model5, type = "response")

ggplot(df, aes(x = x1, y = x2,
               color = predictions,
               pch = y)) +
  geom_point() +
  labs(color = "Predicted Class label",
       pch = "Actual Class Label") +
  ggtitle("Classification of Classes using SVC and Linear Kernel") +
  theme_minimal()
```

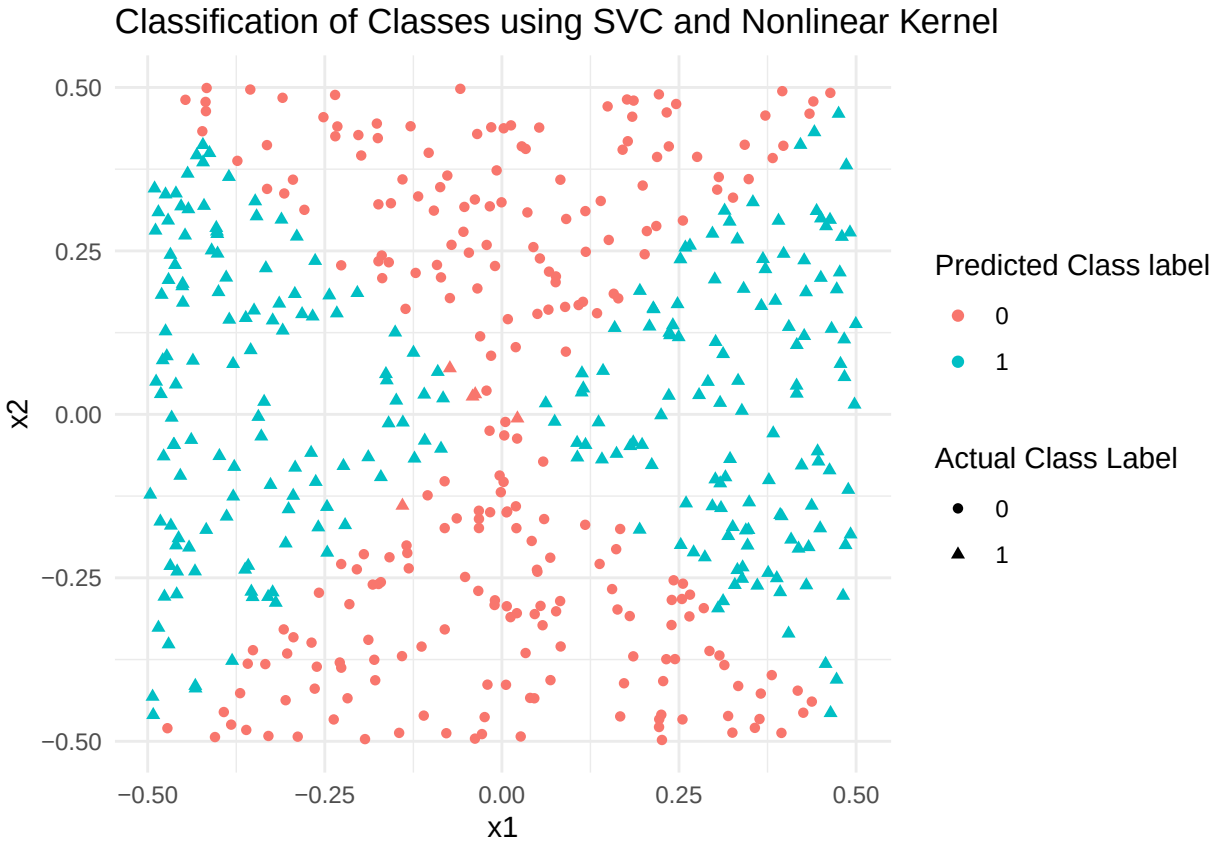



The SVC finds a linear boundary between the two classes and performs numerous misclassifications.

(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*.

```
model6 = svm(y~., df,
             kernel = "polynomial",
             degree = 2)
predictions = predict(model6, type = "response")

ggplot(df, aes(x = x1, y = x2,
               color = predictions,
               pch = y)) +
  geom_point() +
  labs(color = "Predicted Class label",
       pch = "Actual Class Label") +
  ggtitle("Classification of Classes using SVC and Nonlinear Kernel") +
  theme_minimal()
```




(i) Comment on the results.

For the logistic regression models, the confusion matrices for when using a linear approach and nonlinear approach is respectively:

```
table(ifelse(predict(model3) < 0, 0, 1), y)
```

```
##    y
##       0   1
##   0 128  91
##   1 118 163
```

```
table(ifelse(predict(model4) < 0, 0, 1), y)
```

```
##    y
##       0   1
##   0 246   0
##   1   0 254
```

When the nonlinear model is used, no misclassifications are made.

For the SVC/SVM models, the confusion matrices for when using a linear approach and nonlinear approach is respectively:

```
table(predict(model5), y)
```

```
##    y
##       0   1
##   0 127  85
##   1 119 169
```

```
table(predict(model6), y)
```

```
##    y
##       0   1
##   0 246   5
##   1   0 249
```

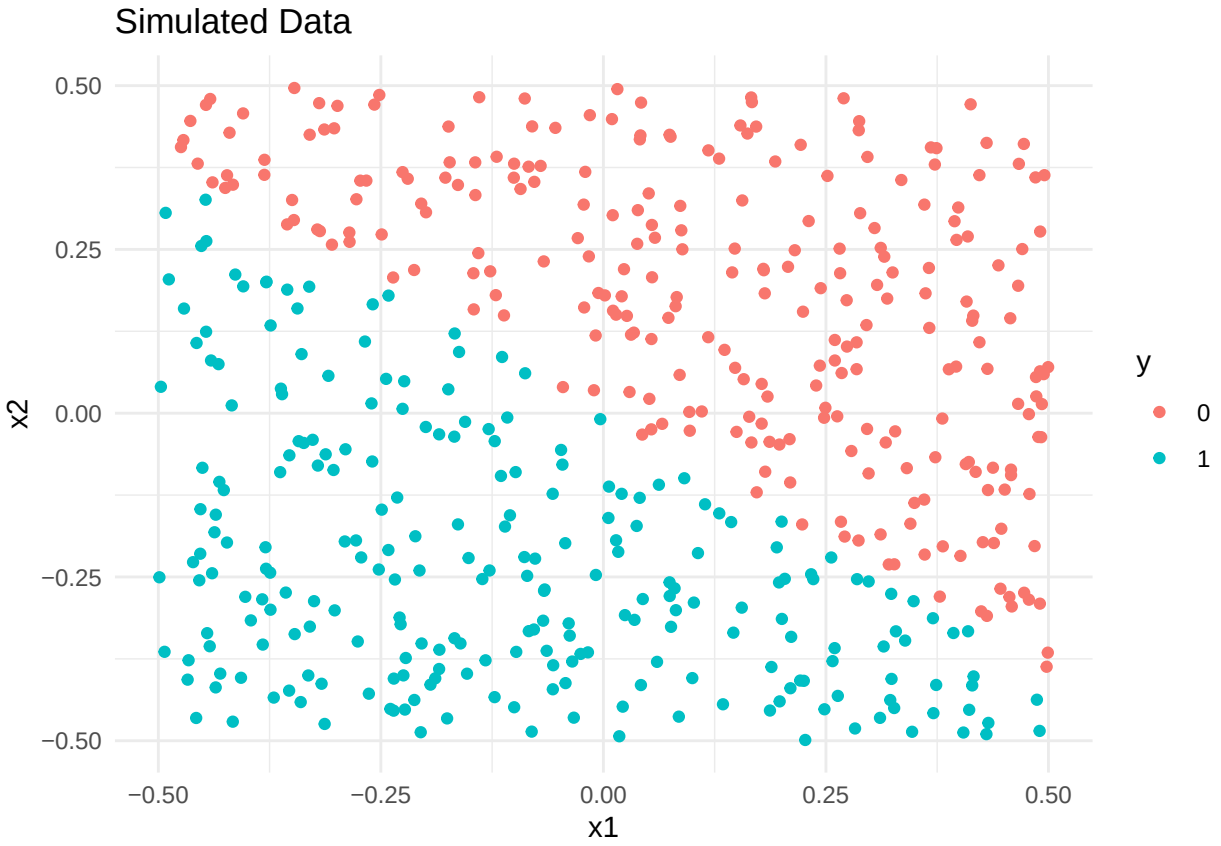
When the nonlinear SVM is used, less misclassifications are made. However, it does not have a 100% classication rate. Therefore, the logistic regression model with nonlinear predictors is the best model to use for binary classification here.

**Question 6: At the end of Section 9.6.1, it is claimed that in the case of data that is just barely linear separable, a support vector classifier with a small value of `cost` that misclassifies a couple of training observations may perform better on test data than one with a huge value of `cost` that does not misclassify any training observations. You will now investigate this claim.**

(a) Generate two-class data with $p = 2$ in such a way that the classes are just barely linearly separable.

```
set.seed(6)
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = ifelse(0.7*x1 + 0.9*x2 > 0, 0, 1)
df = data.frame(x1, x2, y = as.factor(y))

ggplot(df, aes(x = x1, y = x2,
               color = y)) +
  geom_point() +
  ggtitle("Simulated Data") +
  theme_minimal()
```

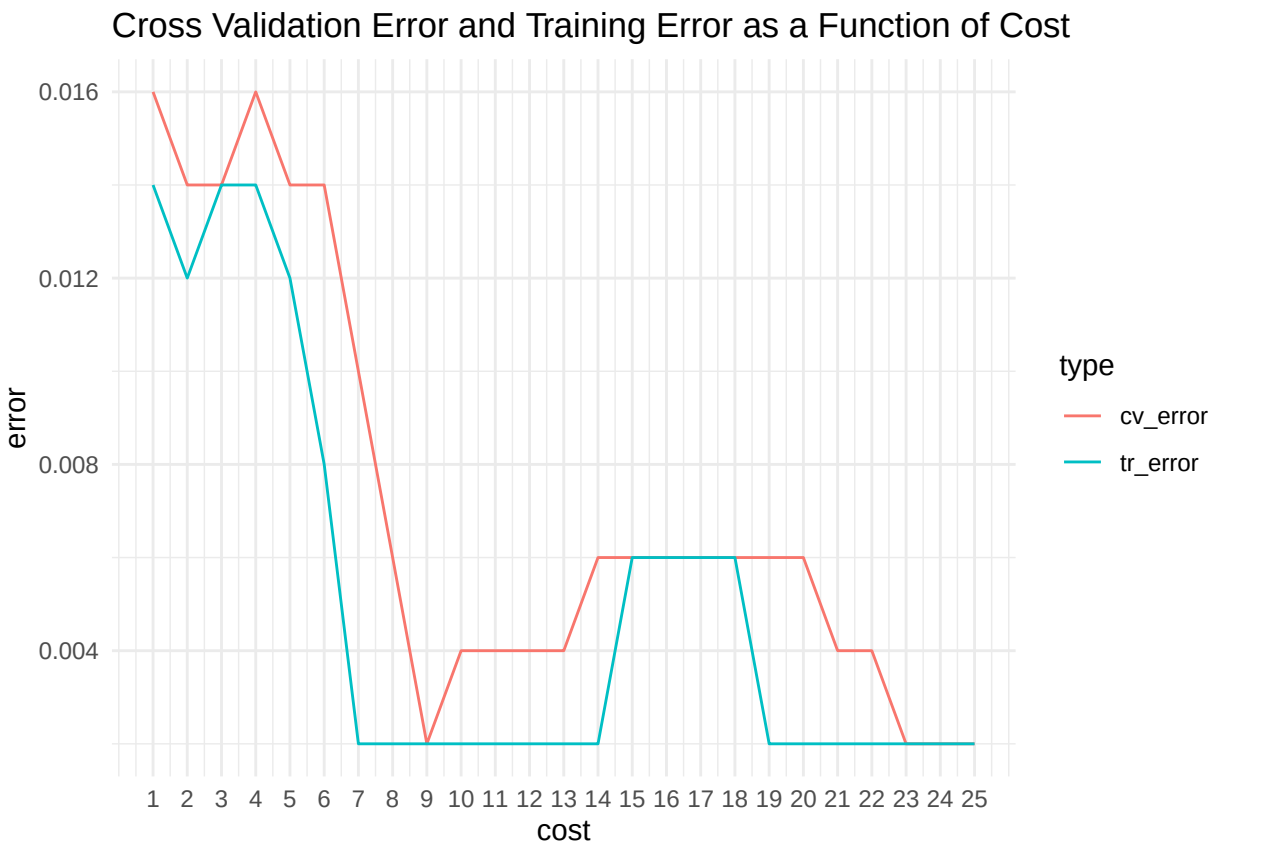



(b) Compute the cross-validation error rates for support vector classifiers with a range of `cost` values. How many training errors are misclassified for each value of `cost` considered, and how does this relate to the cross-validation errors obtained?

```
C = seq(1,25, by = 1)
set.seed(6)
model_tuned = tune(svm, y~., data = df,
                   kernel = "linear",
                   ranges = list(cost = C),
                   tunecontrol = tune.control(cross=25))
cv_error = model_tuned$performances$error
tr_error = c()
for(c in C){
  temp_model = svm(y~., data = df,
                   kernel = "linear",
                   cost = c)
  predictions = predict(temp_model)
  tr_error = c(tr_error, mean(predictions != y))
}
error_df = data.frame(cost = C, cv_error, tr_error)

error_df %>% gather(type, error,
                    cv_error, tr_error) %>%
  ggplot(aes(x = cost, y = error,
             color = type)) +
  geom_path() +
  scale_x_continuous(breaks = C) +
  ggtitle("Cross Validation Error and Training Error as a Function of Cost") +
```

```
  theme_minimal()
```




As the cost increases, the error from cross validation and simply training a SVC with the same cost follow the same trend; if one increases, then the other increases as well, and vice versa. Furthermore, the cross validation error is always more, or the same as training error.

(c) Generate an appropriate test dataset and compute the test errors corresponding to each of the values of `cost` considered. Which value of `cost` leads to the fewest test errors and how does this compare to the values of `cost` that yield the fewest training errors and the fewest cross-validation errors?

```
set.seed(63)
x1_test = runif(100) - 0.5
x2_test = runif(100) - 0.5
y_test = ifelse(0.7*x1 + 0.9*x2 > 0, 0, 1)
df_test = data.frame(x1_test, x2_test,
                     y_test = as.factor(y_test))

test_error = c()
for(c in C){
  temp_model = svm(y~., data = df,
                   kernel = "linear",
                   cost = c)
  predictions = predict(temp_model, df_test)
  test_error = c(test_error,
                 mean(predictions != df_test$y_test))
}
```

The value of `cost` that leads to the fewest test errors is

```
C[which.min(test_error)]
```

```
## [1] 7
```

The value of `cost` that yielded the fewest training error was also 7 while the value of `cost` that yielded the fewest cross-validation errors was 9.

(d) Discuss the results.

Using this dataset, the support vector classifier performed better when utilizing the best `cost` value found using a simple run of the SVC without cross-validation.

## Question 7: In this problem, you will use support vector approaches in order to predict whether a given car gets high or low gas mileage based on the `Auto` dataset.

(a) Create a binary variable that takes on a 1 for cars with gas mileage above the median and a 0 for car mileage below the median.

```
df = Auto
df$mpg = ifelse(df$mpg > median(df$mpg), 1, 0)
```

(b) Fit a support vector classifier to the data with various values of `cost`, in order to predict whether a car gets high or low gas mileage. Report the cross-validation errors associated with different values of this parameter. Comment on the results.

```
C = seq(0.01,100,by = 10)
set.seed(7)
model_tuned_mpg= tune(svm, mpg~., data = df,
                      kernel = "linear",
                      ranges = list(cost = C))
model_tuned_mpg$performances$error
```

```
##  [1] 0.1040393 0.1046226 0.1122701 0.1146626 0.1161324 0.1175254 0.1186349
##  [8] 0.1196113 0.1201102 0.1203954
```

As the `cost` increases, the cross-validation error also increases.

(c) Now repeat(b), this time using SVMs with radial and polynomial basis kernels, with different values of `gamma` and `degree` and `cost`. Comment on the results.

```
model_tuned_mpg_radial = tune(svm, mpg~., data = df,
                              ranges = list(cost = C,
                                            gamma = seq(0,100,
                                                        length = 10)),
                              kernel = "radial")
model_tuned_mpg_poly = tune(svm, mpg~., data = df,
                              ranges = list(cost = C,
                                            degrees = seq(1,100,
                                                          length = 10)),
                              kernel = "polynomial")
```

The best `cost` and `radial` for the SVM with the radial basis kernel is

```
model_tuned_mpg_radial$best.parameters
```

```
##     cost    gamma
## 12 10.01 11.11111
```

The best `cost` and `degree` for the SVM with the polynomial basis kernel is

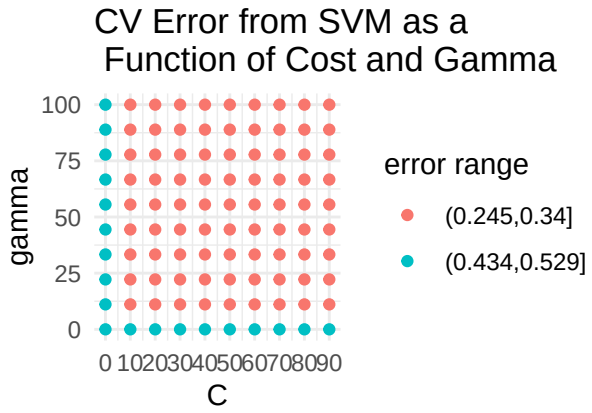
```
model_tuned_mpg_poly$best.parameters
```

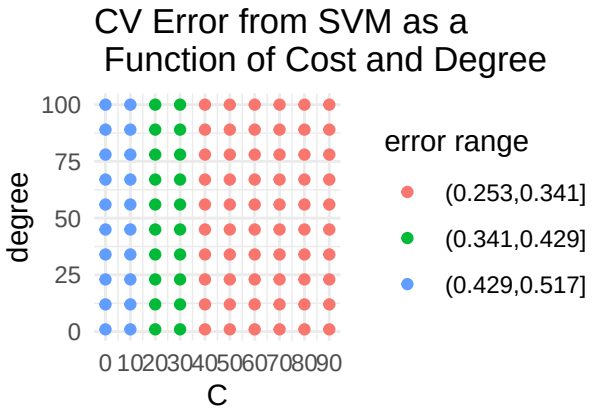
```
##     cost degrees
## 10 90.01       1
```

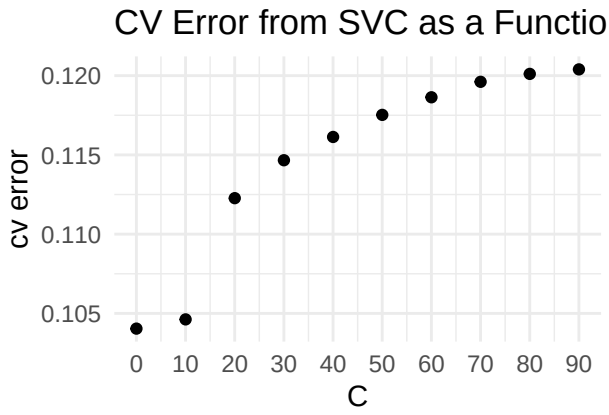
(d) Make some plots to back up the assertions in (b) and (c).

```
error_SVC = data.frame(C = model_tuned_mpg$performances$cost,
                       error = model_tuned_mpg$performances$error)
error_SVM = data.frame(C = model_tuned_mpg_radial$performances$cost,
                       gamma = model_tuned_mpg_radial$performances$gamma,
                       degree = model_tuned_mpg_poly$performances$degrees,
                       error_radial = model_tuned_mpg_radial$performances$error,
                       error_poly = model_tuned_mpg_poly$performances$error)

g1 = ggplot(error_SVC, aes(x = C, y = error)) +
  geom_point() +
  labs(x = "C", y = "cv error") +
  scale_x_continuous(breaks = seq(0,100,
                                  by = 10)) +
  ggtitle("CV Error from SVC as a Function of Cost") +
  theme_minimal()
g2 = ggplot(error_SVM, aes(x = C, y = gamma,
                           color = cut(error_radial, 3))) +
  geom_point() +
  labs(x = "C", y = "gamma",
       color = "error range") +
  scale_x_continuous(breaks = seq(0,100,
                                  by = 10)) +
  ggtitle("CV Error from SVM as a \n Function of Cost and Gamma") +
  theme_minimal()
g3 = ggplot(error_SVM, aes(x = C, y = degree,
                           color = cut(error_poly, 3))) +
  geom_point() +
  labs(x = "C", y = "degree",
       color = "error range") +
  scale_x_continuous(breaks = seq(0,100,
                                  by = 10)) +
  ggtitle("CV Error from SVM as a \n Function of Cost and Degree") +
  theme_minimal()
grid.arrange(g2, g3, g1, nrow = 2)
```




As $C$ increases, cross validation error increases in the SVC model and cross validation error decreases in both SVM models.

### Question 8: This problem involves the `OJ` dataset 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.

```
set.seed(8)
df = OJ
indices = sample(1:nrow(df), size = 800)
train = df[indices,]
test = df[-indices,]
```

(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.

```
model7 = svm(Purchase ~., data = train,
             kernel = "linear", cost = 0.01)
summary(model7)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear",
##     cost = 0.01)
##
```

```
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  436
##
##  ( 218 218 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Using a `cost` of 0.01 and `gamma` value of 0.05, the support vector classifier made a model using 436 support vectors to classify `CH` and `MM`, equal amounts for both classes.

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

The training and test error rates, respectively, are

```
train_svc = mean(predict(model7) != train$Purchase)
test_svc = mean(predict(model7, test) != test$Purchase)
train_svc
```

```
## [1] 0.16625
```

```
test_svc
```

```
## [1] 0.1666667
```

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

```
C = seq(0.01, 10, length = 100)
model7_tuned= tune(svm, Purchase~., data = train,
                   kernel = "linear",
                   ranges = list(cost = C))
```

The best `cost` parameter is

```
model7_tuned$best.parameters
```

```
##        cost
## 32 3.138182
```

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

The training and test error rates, respectively, are

```
train_svc_tuned = mean(predict(model7_tuned$best.model,
                               train) != train$Purchase)
test_svc_tuned = mean(predict(model7_tuned$best.model,
                              test) != test$Purchase)
train_svc_tuned
```

```
## [1] 0.16625
```

```
test_svc_tuned
```

```
## [1] 0.1555556
```

The test error rate is slightly lower than from the previous predefined `cost` parameter SVC model.

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

Let `cost = 0.01`. Fit a SVM with a radial kernel on the training data and report the training and test error rates.

```
model8 = svm(Purchase ~., data = train,
             kernel = "radial", cost = 0.01)
summary(model8)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "radial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  645
##
##  ( 321 324 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

This model created almost the same number of support vectors for both classes. However, it is 1.5 times as many as the linear SVC model.

Using this model, the training and test error rates, respectively, are

```
train_svm_r = mean(predict(model8) != train$Purchase)
test_svm_r = mean(predict(model8, test) != test$Purchase)
train_svm_r
```

```
## [1] 0.40125
```

```
test_svm_r
```

```
## [1] 0.3555556
```

These error rates are greater than the linear SVC error rates. Can it be improved by tuning the hyperparameter `cost`?

When `cost` is optimized, the best `cost` parameter is

```
model8_tuned= tune(svm, Purchase~., data = train,
                   kernel = "radial",
```

```
                    ranges = list(cost = C))
model8_tuned$best.parameters
```

```
##         cost
## 10 0.9181818
```

The training and test error rates, respectively, are

```
train_svm_r_tuned = mean(predict(model8_tuned$best.model,
                                 train) != train$Purchase)
test_svm_r_tuned = mean(predict(model8_tuned$best.model,
                                test) != test$Purchase)
train_svm_r_tuned
```

```
## [1] 0.15
```

```
test_svm_r_tuned
```

```
## [1] 0.1592593
```

These error rates are better than the ones found using the given `cost` value. Furthermore, the training error rate is lower than the optimized linear SVC training error rate. However, the test error rate is slightly higher than the optimized linear SVC test error rate.

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

Let `cost = 0.01`. Fit a SVM with a radial kernel on the training data and report the training and test error rates.

```
model9 = svm(Purchase ~., data = train,
             kernel = "polynomial", degree = 2,
             cost = 0.01)
summary(model9)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "polynomial",
##     degree = 2, cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  648
##
##  ( 321 327 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

This model created almost the same number of support vectors for both classes. However, it is 1.5 times as many as the linear SVC model.

Using this model, the training and test error rates, respectively, are

```
train_svm_p = mean(predict(model9) != train$Purchase)
test_svm_p = mean(predict(model9, test) != test$Purchase)
train_svm_p
```

```
## [1] 0.37125
```

```
test_svm_p
```

```
## [1] 0.362963
```

These error rates are about the same as the SVM with the radial kernel and provided `cost` parameter. Can it be improved by tuning the hyperparameter `cost`?

When `cost` is optimized, the best `cost` parameter is

```
model9_tuned= tune(svm, Purchase~., data = train,
                   kernel = "polynomial",
                   degree = 2,
                   ranges = list(cost = C))
model9_tuned
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  3.743636
##
## - best performance: 0.17875
```

The training and test error rates, respectively, are

```
train_svm_p_tuned = mean(predict(model9_tuned$best.model,
                                 train) != train$Purchase)
test_svm_p_tuned = mean(predict(model9_tuned$best.model,
                                test) != test$Purchase)
train_svm_p_tuned
```

```
## [1] 0.16
```

```
test_svm_p_tuned
```

```
## [1] 0.1592593
```

The test error rate calculated here is the same as the one found using a radial kernel with an optimize `cost` parameter.

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

```
error_df = data.frame(train = c(train_svc, train_svc_tuned,
                                train_svm_p, train_svm_p_tuned,
                                train_svm_r, train_svm_r_tuned),
                      test = c(test_svc, test_svc_tuned,
                               test_svm_p, test_svm_p_tuned,
```

```
                                test_svm_r, test_svm_r_tuned))
rownames(error_df) = c("SVC", "SVC Tuned", "SVM Polynomial",
                       "SVM Polynomial Tuned", "SVM Radial",
                       "SVM Radial Tuned")
error_df
```

```
##                         train      test
## SVC                   0.16625 0.1666667
## SVC Tuned             0.16625 0.1555556
## SVM Polynomial        0.37125 0.3629630
## SVM Polynomial Tuned 0.16000 0.1592593
## SVM Radial            0.40125 0.3555556
## SVM Radial Tuned      0.15000 0.1592593
```

The best results on this data is given by the tuned support vector classifier.

All of the practice applied exercises in this document are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.