MLStats: Support Vector Machines

Darshan Patel

3/4/2019

In this assignment, mimic the lab exercises from ISLR Chapter 9: Support Vector Machines.

Libraries

Load the following libraries.

```
rm(list = ls())
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(ROCR)
```

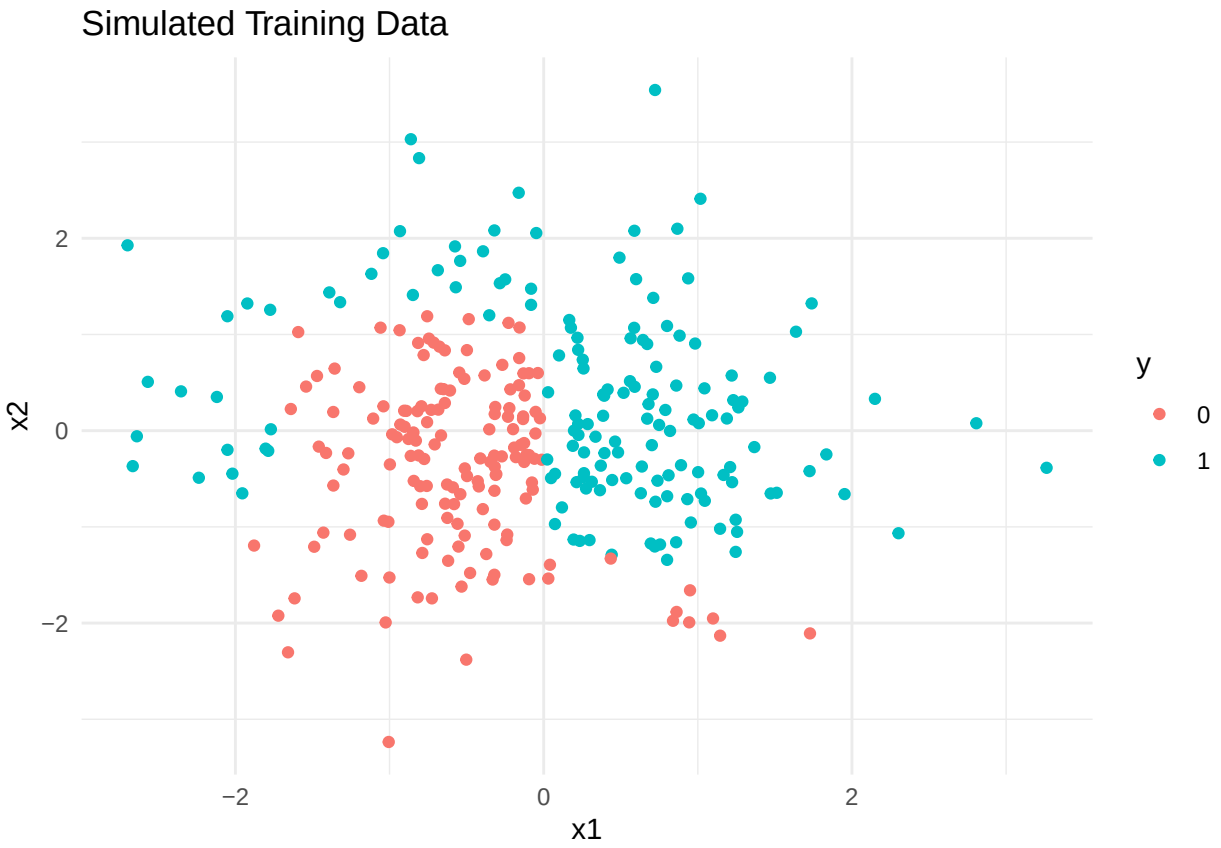
```
## Warning: package 'gplots' was built under R version 3.4.4
```

Dataset

In this assignment, the dataset that will be used will be simulated. The reason for this is so various types of decision boundaries can be explored on numeric predictors that have scattered values and that plots can be made to show decision boundaries and the effect of using different kernels.

First simulate a random dataset with 2 predictors and 2 classes and create a train and test split. Plot the training data.

```
set.seed(2019)
x1 = rnorm(500)
x2 = rnorm(500)
y = as.factor(ifelse(0.6*x2^5 - 0.5*x2^3 + 1.2*x1^2 + 2*x1 >  0, 1, 0))
df = data.frame(x1, x2, y)
indices = sample(1:nrow(df), size = 0.6*nrow(df))
train = df[indices,]
test = df[-indices,]
ggplot(train, aes(x = x1, y = x2, color = y)) + geom_point() +
  ggtitle("Simulated Training Data") +
  theme_minimal()
```



This dataset has 2 predictors, `x1` and `x2` that classifies `y` into one of two classes. The distinction between the two classes cannot be made using a linear decision boundary based on the training data. The data is not linearly separable. Let's see what happens if we try to make a linear boundary.

Support Vector Classifier

Fit a support vector classifier on the training data to perform classification.

```
model_svc = svm(y~., data = train,
                kernel = "linear")
summary(model_svc)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.5
##
## Number of Support Vectors:  165
##
##  ( 82 83 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

This linear support vector classifier model has 165 support vectors, when the `cost` is not provided as a user-defined parameter. Both classes get about the same number of support vectors.

The confusion matrix when predicting on the training data is

```
table(predictions = predict(model_svc),
      actual = train$y)
```

```
##            actual
## predictions   0   1
##           0 125  28
##           1  24 123
```

Many classifications are successfully made but several errors are also made. This makes sense; the training data does not appear to be linearly separable, thus a clear decision boundary cannot be made. How about on the test data?

The confusion matrix when predicting on the test data is

```
table(predictions = predict(model_svc, test),
      actual = test$y)
```

```
##            actual
## predictions  0  1
##           0 84 16
##           1 17 83
```

Many misclassifications are made. 33!

The error rate is

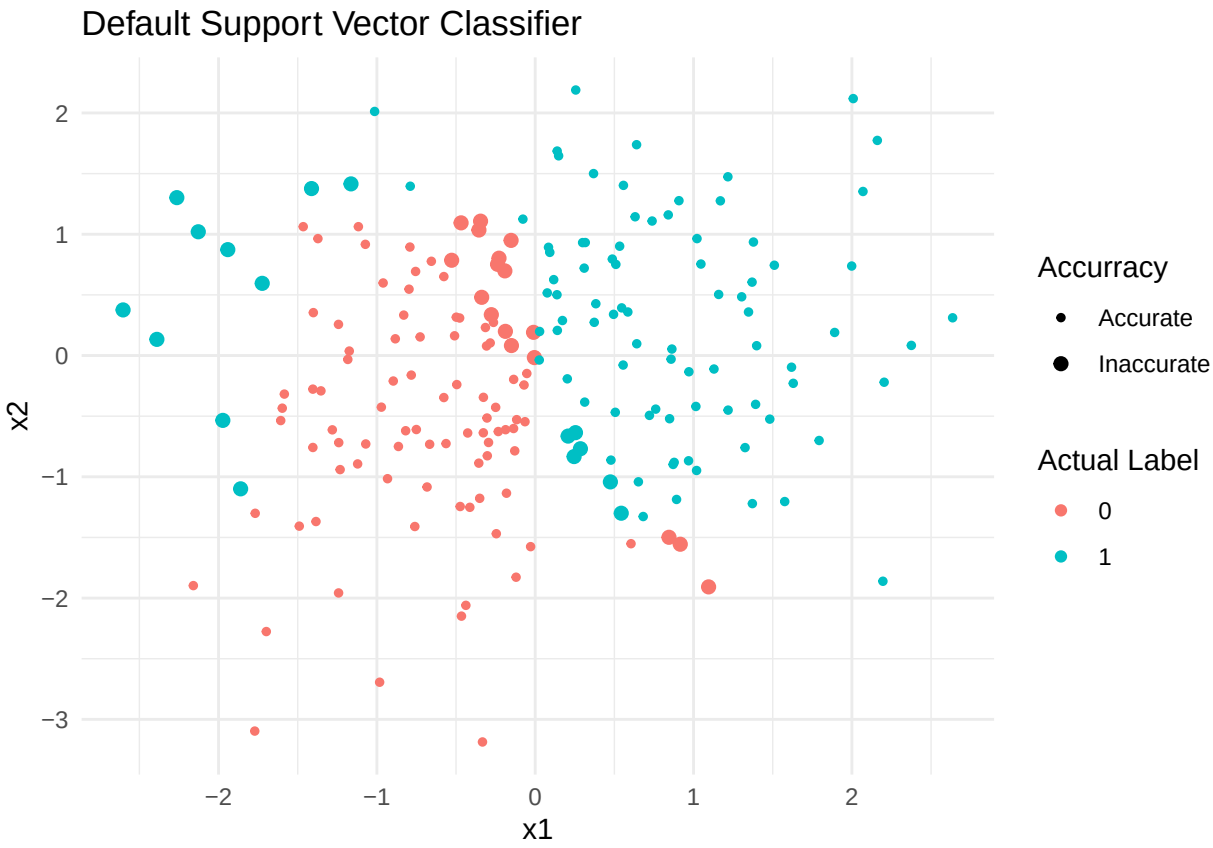
```
mean(predict(model_svc, test) != test$y)
```

```
## [1] 0.165
```

The support vector classifier can be plotted on the test set.

```
ggplot(test, aes(x = x1, y = x2, color = y)) +
  geom_point(aes(size = predict(model_svc, test) != y)) +
  labs(color = "Actual Label", size = "Accuracy") +
  scale_size_discrete(name="Accurracy",
                      range = c(1, 2),
                      labels=c("Accurate", "Inaccurate")) +
  ggtitle("Default Support Vector Classifier") +
  theme_minimal()
```

```
## Warning: Using size for a discrete variable is not advised.
```



As can be seen in this plot, this model made some misclassifications near the boundary and some at the farther ends.

What happens when `cost` goes up? Look at the confusion matrix.

```
model_svc_more_cost = svm(y~., data = train,
                          kernel = "linear", cost = 100)
table(predictions = predict(model_svc_more_cost, test),
      actual = test$y)
```

```
##            actual
## predictions  0  1
##           0 86 17
##           1 15 82
```

By increasing `cost`, the number of misclassifications went slightly down.

What happens when `cost` goes down?

```
model_svc_less_cost = svm(y~., data = train,
                          kernel = "linear", cost = 0.01)
table(predictions = predict(model_svc_less_cost, test),
      actual = test$y)
```

```
##            actual
## predictions  0  1
##           0 90 19
##           1 11 80
```

The number of misclassifications went down when `cost` went down.

To find the best `cost` parameter to use, perform cross-validation on the dataset.

```
set.seed(1)
model_svc_tune = tune(svm, y~., data = train,
                      kernel = "linear",
                      ranges = list(cost = seq(0.01, 100,
                                               length = 100)))
```

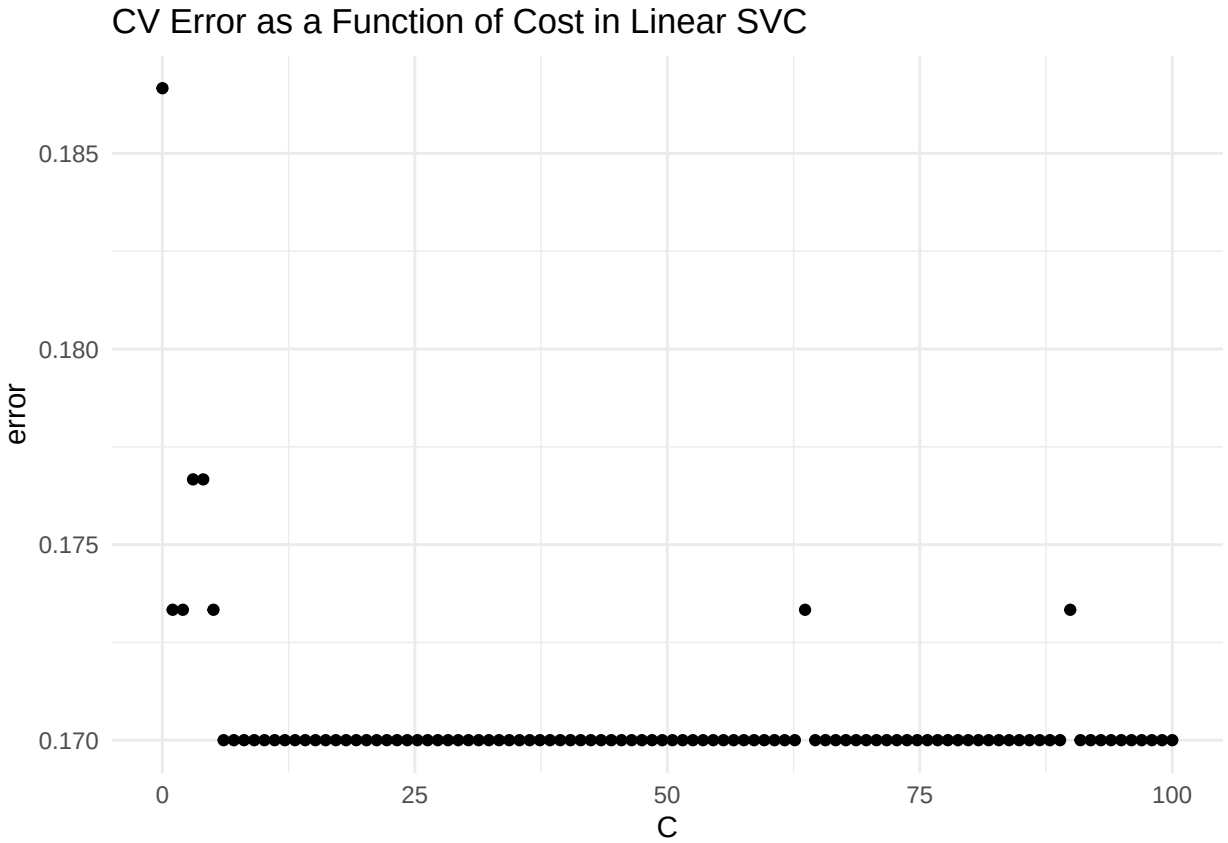
The best `cost` parameter is

```
model_svc_tune$best.parameters
```

```
##   cost
## 7 6.07
```

The error can also be plotted as a function of `cost`.

```
error_cv_df = data.frame(C = model_svc_tune$performances$cost,
                         error = model_svc_tune$performances$error)
ggplot(error_cv_df, aes(x = C, y = error)) +
  geom_point() +
  ggtitle("CV Error as a Function of Cost in Linear SVC") +
  theme_minimal()
```



In this cross validation run, the cross validation error achieved a global minimum far before C could get very large and then stayed constant with occasional jumps.

Using this best value of `cost`, predict on the test dataset. The test error rate is

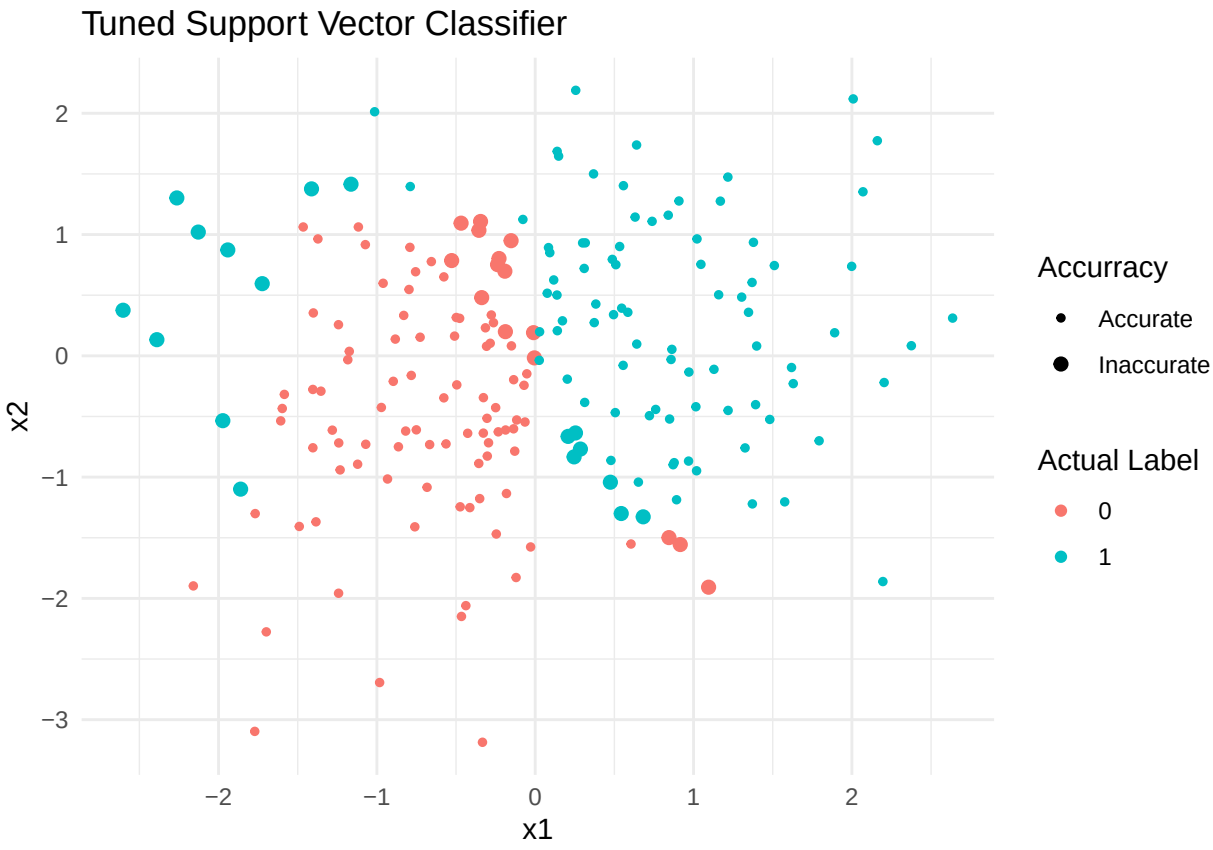
```
mean(predict(model_svc_tune$best.model, test) != test$y)
```

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

With the best `cost` parameter value used, the test set error rate is 0.16. This is slightly lower than the one found using the default `cost` parameter of 1. The plot of the classifier on the test data is shown below.

```
ggplot(test, aes(x = x1, y = x2, color = y)) +
  geom_point(aes(size = predict(model_svc_tune$best.model, test) != y)) +
  labs(color = "Actual Label", size = "Accuracy") +
  scale_size_discrete(name="Accurracy",
                      range = c(1, 2),
                      labels=c("Accurate", "Inaccurate")) +
  ggtitle("Tuned Support Vector Classifier") +
  theme_minimal()
```

```
## Warning: Using size for a discrete variable is not advised.
```



The confusion matrix is

```
table(predictions = predict(model_svc_tune$best.model, test),
      actual = test$y)
```

```
##            actual
## predictions  0  1
##           0 86 17
##           1 15 82
```

This model only made 1 less misclassification than the support vector classifier using the default `cost` parameter on the test set.

Try improving using non-linear kernels.

Support Vector Machine

Fit the training data using a radial kernel for the SVM with a default γ parameter of 1 and `cost` of 1.

```
model_svm_rad = svm(y~., data = train,
                    kernel = "radial", gamma = 1)
summary(model_svm_rad)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "radial", gamma = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  1
##
## Number of Support Vectors:  82
##
##  ( 43 39 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

This model uses 82 support vectors in a not even split between the two classes.

The test error rate is

```
mean(predict(model_svm_rad, test) != test$y)
```

```
## [1] 0.04
```

The confusion matrix is

```
table(predictions = predict(model_svm_rad, test),
      actual = test$y)
```

```
##            actual
## predictions  0  1
##           0 98  5
##           1  3 94
```

Errors are almost 0. Only 8 errors were made.

Perform cross validation to find the best choice of γ and `cost`.

```
set.seed(1)
model_svm_rad_tune = tune(svm, y~., data = train,
                          kernel = "radial",
                          ranges = list(cost = c(0.1, 1, 10,
                                                 100, 1000),
                                        gamma = seq(0.01, 1,
                                                    length = 100)))
model_svm_rad_tune$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, ranges = list(cost = c(0.1,
##     1, 10, 100, 1000), gamma = seq(0.01, 1, length = 100)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  100
##       gamma:  0.36
##
## Number of Support Vectors:  25
```

The best support vector machine with a radial kernel has a `cost` of 100 and γ value of 0.36.

Using these values, the test set error is

```
mean(predict(model_svm_rad_tune$best.model, test) != test$y)
```

```
## [1] 0.025
```

and the confusion matrix is

```
table(predictions = predict(model_svm_rad_tune$best.model, test),
      actual = test$y)
```

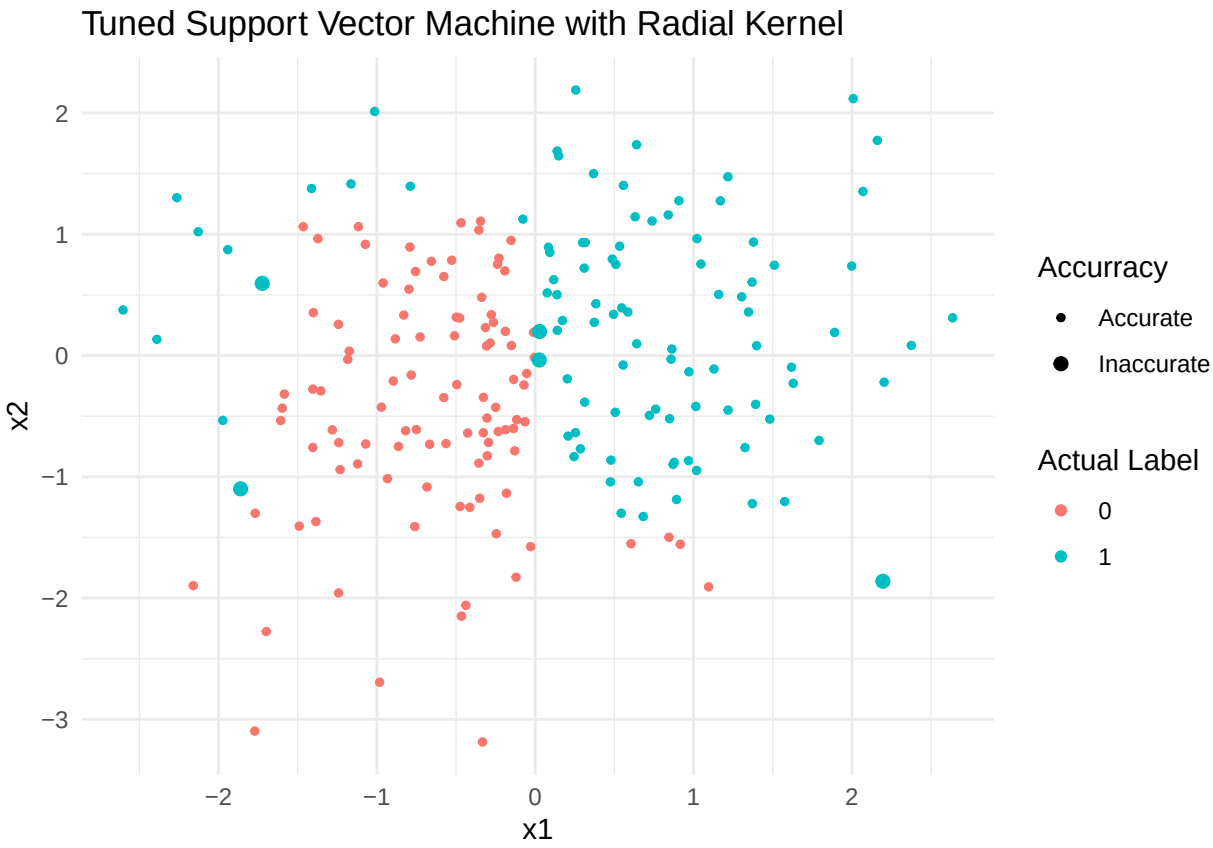
```
##            actual
## predictions   0   1
##           0 101   5
##           1   0  94
```

Only 5 misclassifications are made!

This plot is plotted on the test set.

```
ggplot(test, aes(x = x1, y = x2, color = y)) +
  geom_point(aes(size = predict(model_svm_rad_tune$best.model, test) != y)) +
  labs(color = "Actual Label", size = "Accuracy") +
  scale_size_discrete(name="Accurracy",
                      range = c(1, 2),
                      labels=c("Accurate", "Inaccurate")) +
  ggtitle("Tuned Support Vector Machine with Radial Kernel") +
  theme_minimal()
```

```
## Warning: Using size for a discrete variable is not advised.
```



Misclassifications are still made near the boundaries.

Let's try now with a polynomial kernel.

Fit the training data using a polynomial kernel for the SVM with a default `degree` parameter of 2 and `cost` of 1.

```
model_svm_poly = svm(y~., data = train,
                     kernel = "polynomial", degree = 2)
summary(model_svm_poly)
```

```
##
## Call:
## svm(formula = y ~ ., data = train, kernel = "polynomial", degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##       gamma:  0.5
##      coef.0:  0
##
## Number of Support Vectors:  236
##
##  ( 118 118 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  0 1
```

This model uses 236 support vectors in an even split between the two classes. This is nearly 3 times as many as the radial kernel SVM!

The test error rate is

```
mean(predict(model_svm_poly, test) != test$y)
```

```
## [1] 0.35
```

A lot of errors are made.

The confusion matrix is

```
table(predictions = predict(model_svm_poly, test),
      actual = test$y)
```

```
##            actual
## predictions  0  1
##           0 81 50
##           1 20 49
```

Too many errors. More than half of the 1's got misclassified.

Perform cross validation to find the best choice of `degree` and `cost`.

```
set.seed(1)
model_svm_poly_tune = tune(svm, y~., data = train,
                           kernel = "polynomial",
                          ranges = list(cost = c(0.1, 1, 10,100),
                                        degree = c(2, 3, 4, 5)))
model_svm_poly_tune$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, ranges = list(cost = c(0.1,
##     1, 10, 100), degree = c(2, 3, 4, 5)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##       gamma:  0.5
##      coef.0:  0
##
## Number of Support Vectors:  218
```

The best support vector machine with a polynomial kernel has a `cost` of 10 and `degree` of 3. It has 218 support vectors.

Using these values, the test set error is

```
mean(predict(model_svm_poly_tune$best.model, test) != test$y)
```

```
## [1] 0.23
```

and the confusion matrix is

```
table(predictions = predict(model_svm_poly_tune$best.model, test),
      actual = test$y)
```

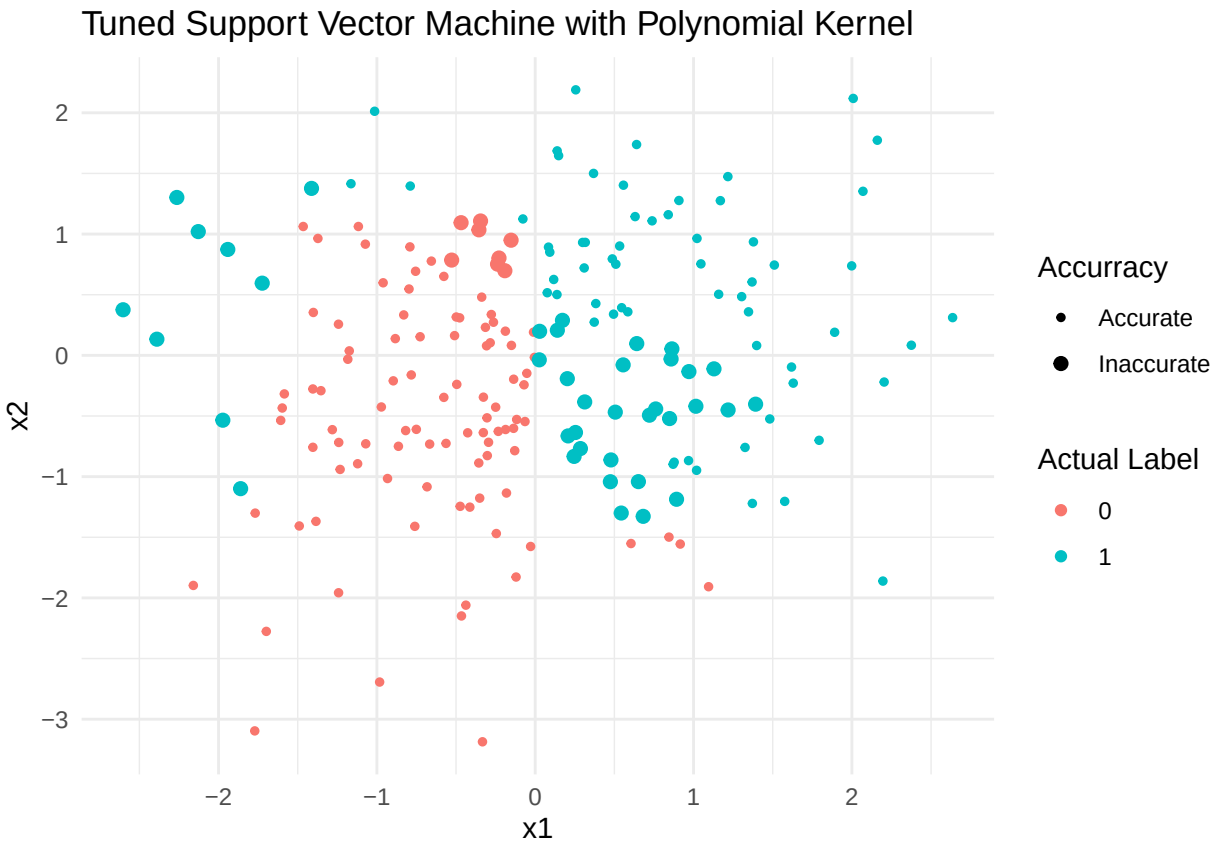
```
##            actual
## predictions  0  1
##           0 93 38
##           1  8 61
```

46 misclassifications are made! This is very bad.

This plot is plotted on the test set.

```
ggplot(test, aes(x = x1, y = x2, color = y)) +
  geom_point(aes(size = predict(model_svm_poly_tune$best.model, test) != y)) +
  labs(color = "Actual Label", size = "Accuracy") +
  scale_size_discrete(name="Accurracy",
                      range = c(1, 2),
                      labels=c("Accurate", "Inaccurate")) +
  ggtitle("Tuned Support Vector Machine with Polynomial Kernel") +
  theme_minimal()
```

```
## Warning: Using size for a discrete variable is not advised.
```



What is apparent in this plot is that even though the 1s are in a neighborhood of its only class only, they are still misclassified.

For this classification problem, the radial kernel with tuned paramter performed the best on the test data.

ROC Curve

Plot the ROC curve using the training data and test data.

First create a function to plot an ROC curve given a vector containing a score for each observation, `pred` and a vector containing the class label for each observation, `truth`.

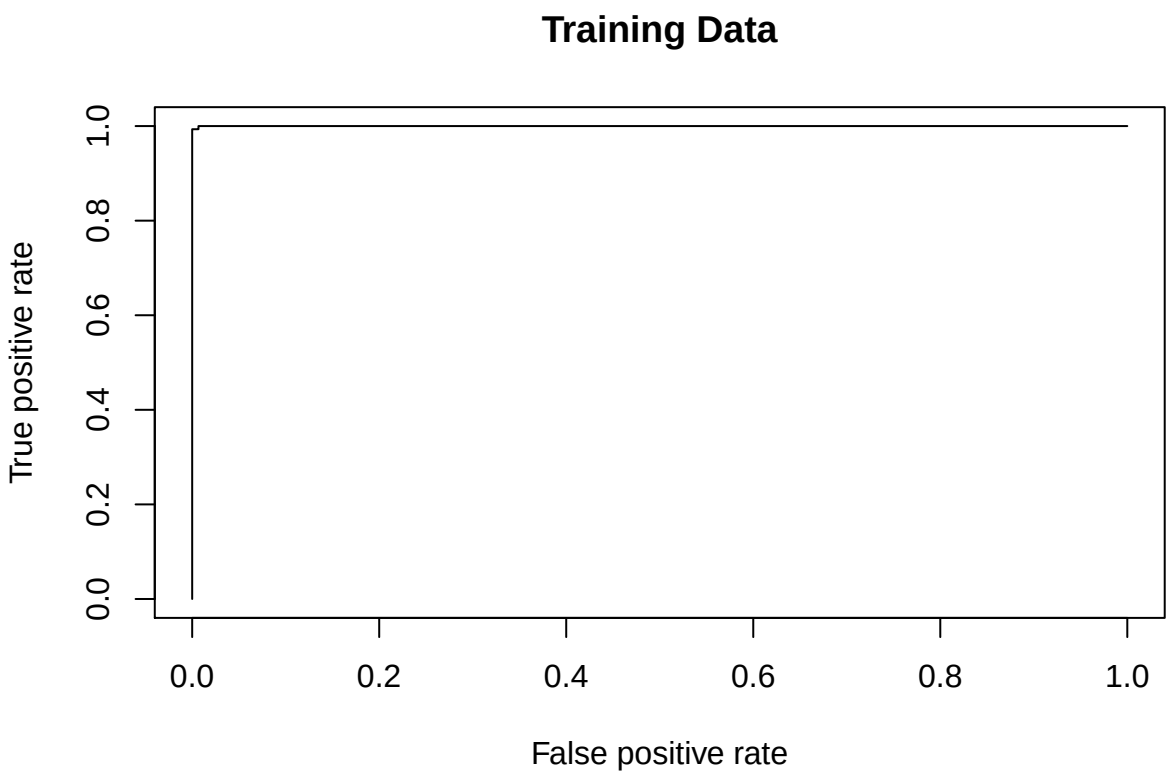
```
rocplot = function(pred, truth, ...){
  predob = prediction(pred, truth)
  perf = performance(predob, "tpr", "fpr")
  plot(perf, ...)
}
```

Now create an SVM using our best model, with radial kernel with $\gamma = 100$ and `cost` $= 0.36$.

```
roc_model = svm(y~., data = train, kernel = "radial",
                gamma = 100, cost = 0.36, decision.values = TRUE)
fitted_vals = attributes(predict(roc_model,
                                 train,
                                 decision.values = TRUE))$decision.values
```

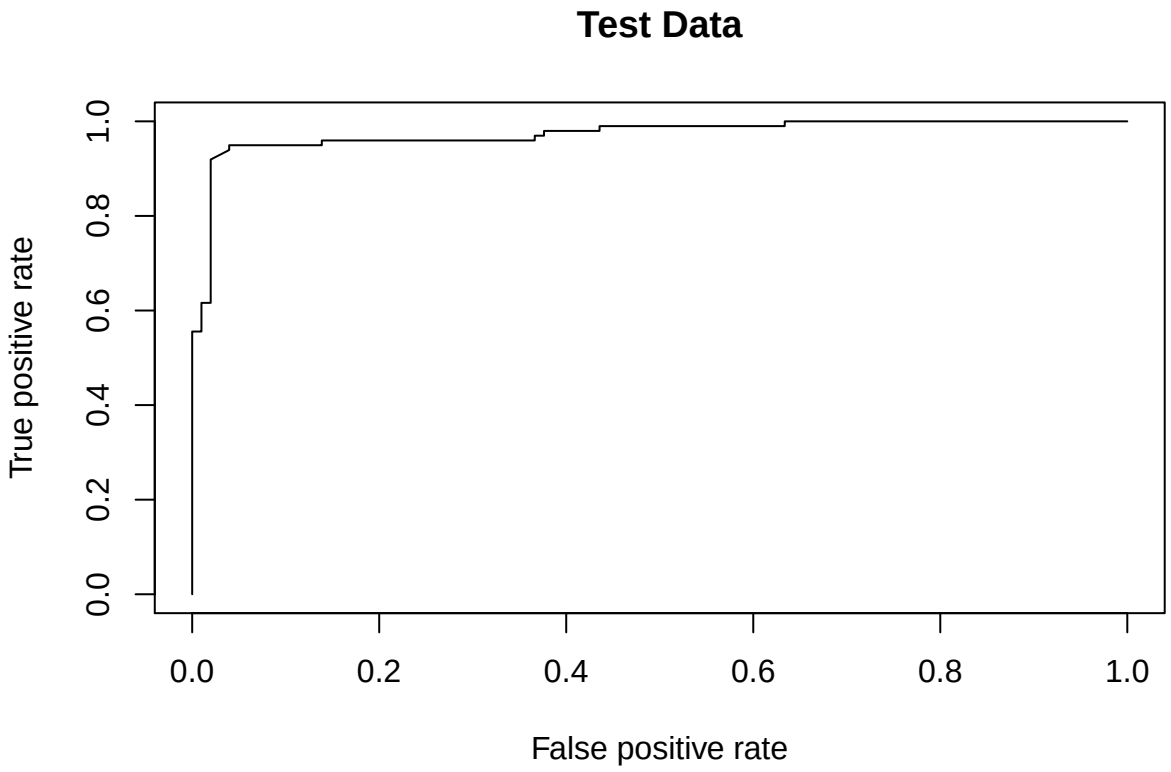
Now produce the ROC plot.

```
rocplot(fitted_vals, train$y,
        main = "Training Data")
```



The SVM does appear to make accurate predictions. Check on the test data.

```
fitted_vals = attributes(predict(roc_model,
                                 test,
                                 decision.values = TRUE))$decision.values
rocplot(fitted_vals, test$y,
        main = "Test Data")
```



There is a more inward behavior of the curve than before. However, the classifier is still hugging the top left corner and so the classifier is optimal.

SVM with Multiple Class

Create a dataset with 4 features and a train/test split. Plot the training data.

```
set.seed(2019)
x1 = rnorm(500)
x2 = rnorm(500)
y = as.factor(cut(0.02*x1^3 + 0.1*x2^2, 4,
                  labels = c(0, 1, 2, 3)))
df = data.frame(x1, x2, y)
indices = sample(1:nrow(df), size = 0.6*nrow(df))
train = df[indices,]
test = df[-indices,]
ggplot(train, aes(x = x1, y = x2, color = y)) +
  geom_point() +
  ggtitle("Simulated Training Data with 4 Classes") +
  theme_minimal()
```

The boundaries between classes here are nonlinear.

First fit a linear SVC to the training data using default configurations.

```
class_4_model_svc = svm(y~., data = train,
                        kernel = "linear")
```

The test error rate is

```
mean(predict(class_4_model_svc, test) != test$y)
```

```
## [1] 0.395
```

This is a high error rate.

The confusion matrix is shown below.

```
table(predictions = predict(class_4_model_svc, test),
      actual = test$y)
```

```
##            actual
## predictions  0  1  2  3
##           0 62 47  3  1
##           1 24 59  4  0
##           2  0  0  0  0
##           3  0  0  0  0
```

A lot of `0`'s and `1`'s got misclassified. This makes sense; in the plot of the training data, there does not appear to be linear decision boundary between the two.

Try polynomial kernel of degree `2` and report the test error rate.

```
model_4_model_svm_poly = svm(y~., data = train,
                             kernel = "polynomial", degree = 2)
mean(predict(model_4_model_svm_poly, test) != test$y)
```

```
## [1] 0.19
```

The test error rate went down when a polynomial kernel of degree 2 was used.

The confusion matrix is

```
table(predictions = predict(model_4_model_svm_poly, test),
      actual = test$y)
```

```
##            actual
## predictions  0  1  2  3
##           0 81 29  0  0
##           1  5 76  2  0
##           2  0  1  4  0
##           3  0  0  1  1
```

There is a great reduction in the misclassifications of 0's and 1's. Furthermore, the single 3 value got classified correctly now, as well as most of the 2's.

Can the radial kernel improve classification? Try the radial kernel with $\gamma = 1$ and report the test error rate.

```
model_4_model_svm_rad = svm(y~., data = train,
                            kernel = "radial", gamma = 1)
mean(predict(model_4_model_svm_rad, test) != test$y)
```

```
## [1] 0.085
```

This is the lowest test error rate found for the classification of 4 classes.

The confusion matrix is

```
table(predictions = predict(model_4_model_svm_rad, test),
      actual = test$y)
```

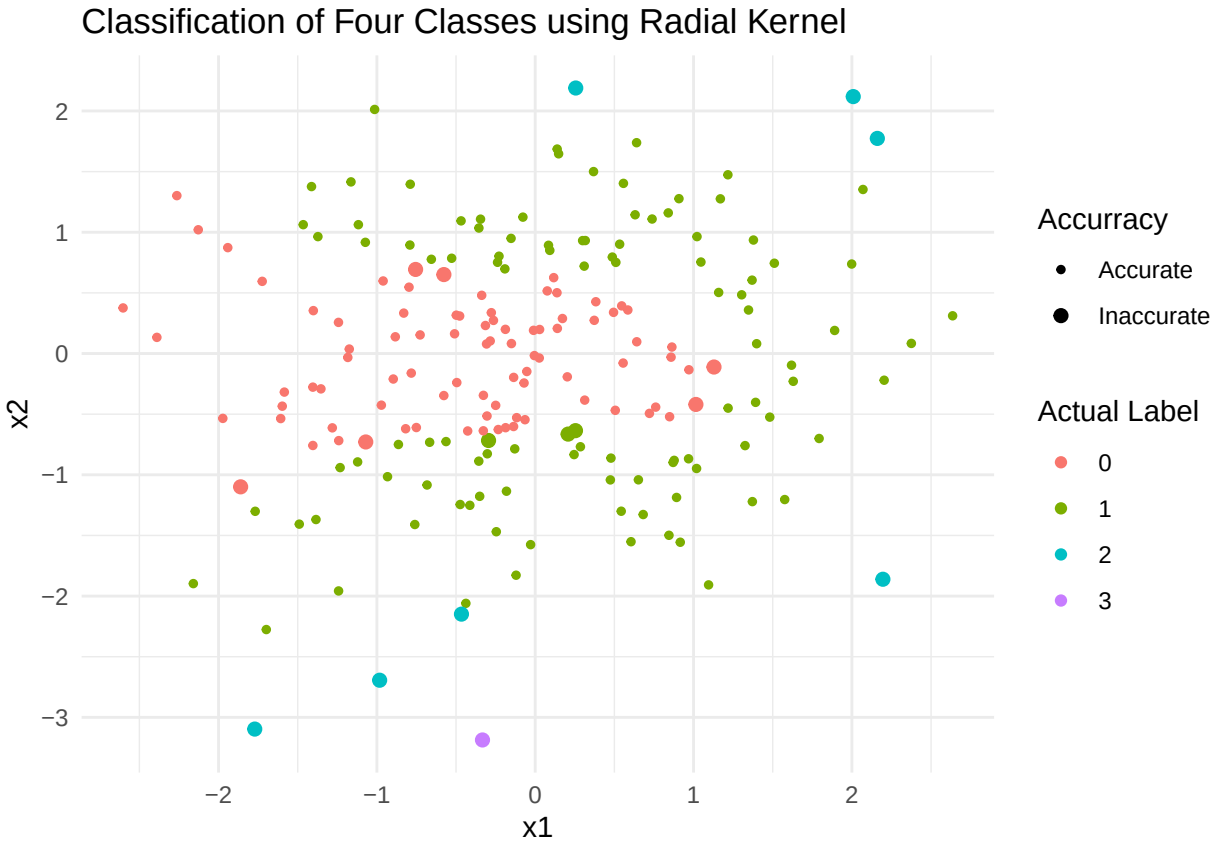
```
##            actual
## predictions   0   1   2   3
##           0  80   3   0   0
##           1   6 103   7   1
##           2   0   0   0   0
##           3   0   0   0   0
```

The low test error rate comes with a fallback; the small number of 2's and 3s all got misclassified when using the radial kernel model.

A plot of the radial kernel SVM classifier is plotted on the test data below.

```
ggplot(test, aes(x = x1, y = x2, color = y)) +
  geom_point(aes(size = predict(model_4_model_svm_rad, test) != y)) +
  labs(color = "Actual Label", size = "Accuracy") +
  scale_size_discrete(name="Accurracy",
                      range = c(1, 2),
                      labels=c("Accurate", "Inaccurate")) +
  labs(color = "Actual Label", size = "Accuracy") +
  ggtitle("Classification of Four Classes using Radial Kernel") +
  theme_minimal()
```

```
## Warning: Using size for a discrete variable is not advised.
```



It appears to be that the classifier couldn't classify the single `3` class value at the bottom of the plot, as well as all the `2`s that had a value of `x2` less than 0. The `1`s that got misclassified are very close to the `0`s and vice versa.

In my opinion, the radial kernel for the SVM is the best classifier for the 4 class classification problem. Additional improvements can be made by tuning the parameters.

All of the lab instructions 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.