Generating a radially separable dataset

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

Generating a 2d uniformly distributed set of points

- Generate a dataset with 200 points
  - 2 predictors `x1` and `x2`, uniformly distributed between -1 and 1.

```
# Set required number of datapoints
n <- 200
# Set seed to ensure reproducibility
set.seed(42)

# Generate dataframe with 2 predictors x1 and x2 in (-1, 1)
df <- data.frame(x1 = runif(n, min = -1, max = 1),
                 x2 = runif(n, min = -1, max = 1))
```

Create a circular boundary

- Create a circular decision boundary of radius 0.7 units.
- Categorical variable y is +1 or -1 depending on the point lies outside or within boundary.

```
radius <- 0.7
radius_squared <- radius ^ 2

#categorize data points depending on location wrt boundary
df$y <- factor(ifelse(df$x1 ^ 2 + df$x2 ^ 2 < radius_squared, -1, 1),
               levels = c(-1, 1))
```

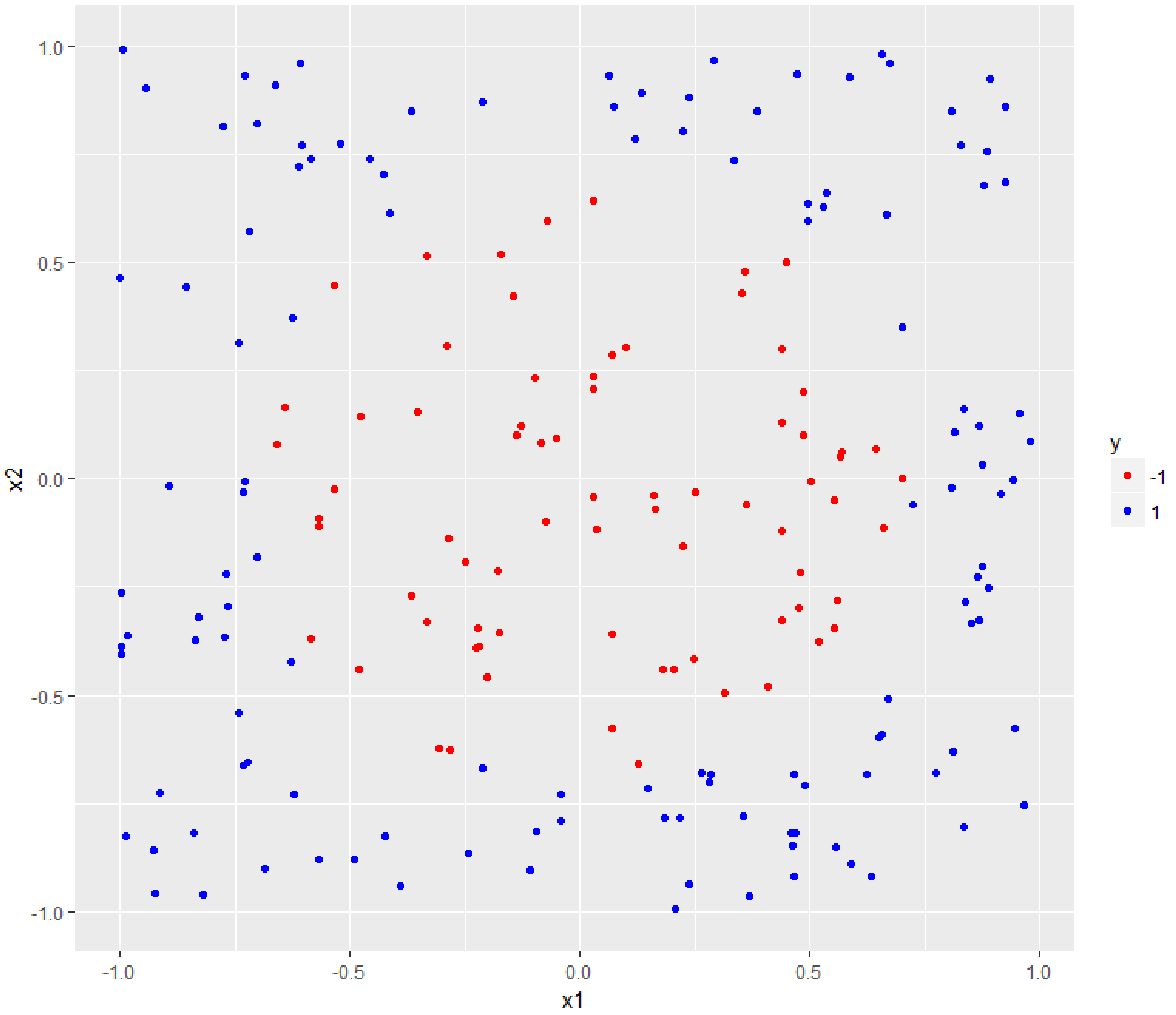
Plot the dataset

- Visualize using ggplot.

```
library(ggplot2)
```

- predictors plotted on 2 axes; classes distinguished by color.

```
# Build plot
p <- ggplot(data = df, aes(x = x1, y = x2, color = y)) +
     geom_point() +
     scale_color_manual(values = c("-1" = "red", "1" = "blue"))
# Display plot
p
```


1.0
0.5
0.0
-0.5
-1.0
x2
-1.0
-0.5
0.0
0.5
1.0
x1
y
-1
1

Adding a circular boundary - Part 1

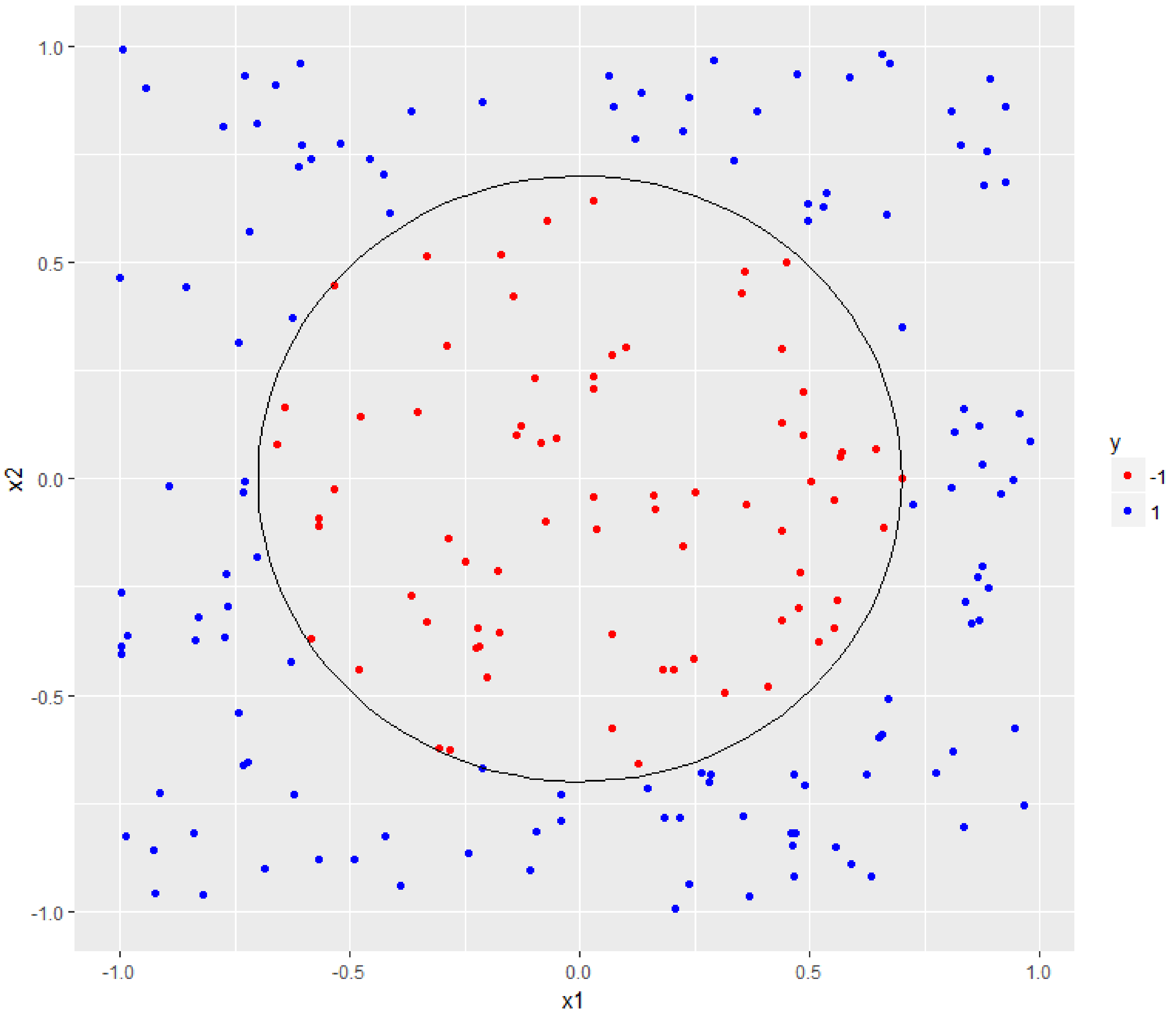
- We'll create a function to generate a circle

```
# Function generates dataframe with points
# lying on a circle of radius r
circle <-
  function(x1_center, x2_center, r, npoint = 100) {

  # Angular spacing of 2*pi/npoint between points
  theta <- seq(0, 2 * pi, length.out = npoint)
  x1_circ <- x1_center + r * cos(theta)
  x2_circ <- x2_center + r * sin(theta)

  data.frame(x1c = x1_circ, x2c = x2_circ)
}
```

Adding a circular boundary - Part 2

- To add boundary to plot:
  - generate boundary using `circle()` function.
  - add boundary to plot using `geom_path()`

```
# Generate boundary
boundary <- circle(x1_center = 0,
                   x2_center = 0,
                   r = radius)
# Add boundary to previous plot
p <- p +
     geom_path(data = boundary,
               aes(x = x1c, y = x2c),
               inherit.aes = FALSE)
# Display plot
p
```


1.0
0.5
0.0
-0.5
-1.0
x2
-1.0
-0.5
0.0
0.5
1.0
x1
y
-1
1

Time to practice!

SUPPORT VECTOR MACHINES IN R

Linear SVMs on radially separable data

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

Linear SVM, cost = 1

- Partition radially separable dataset into training/test (seed = 10)

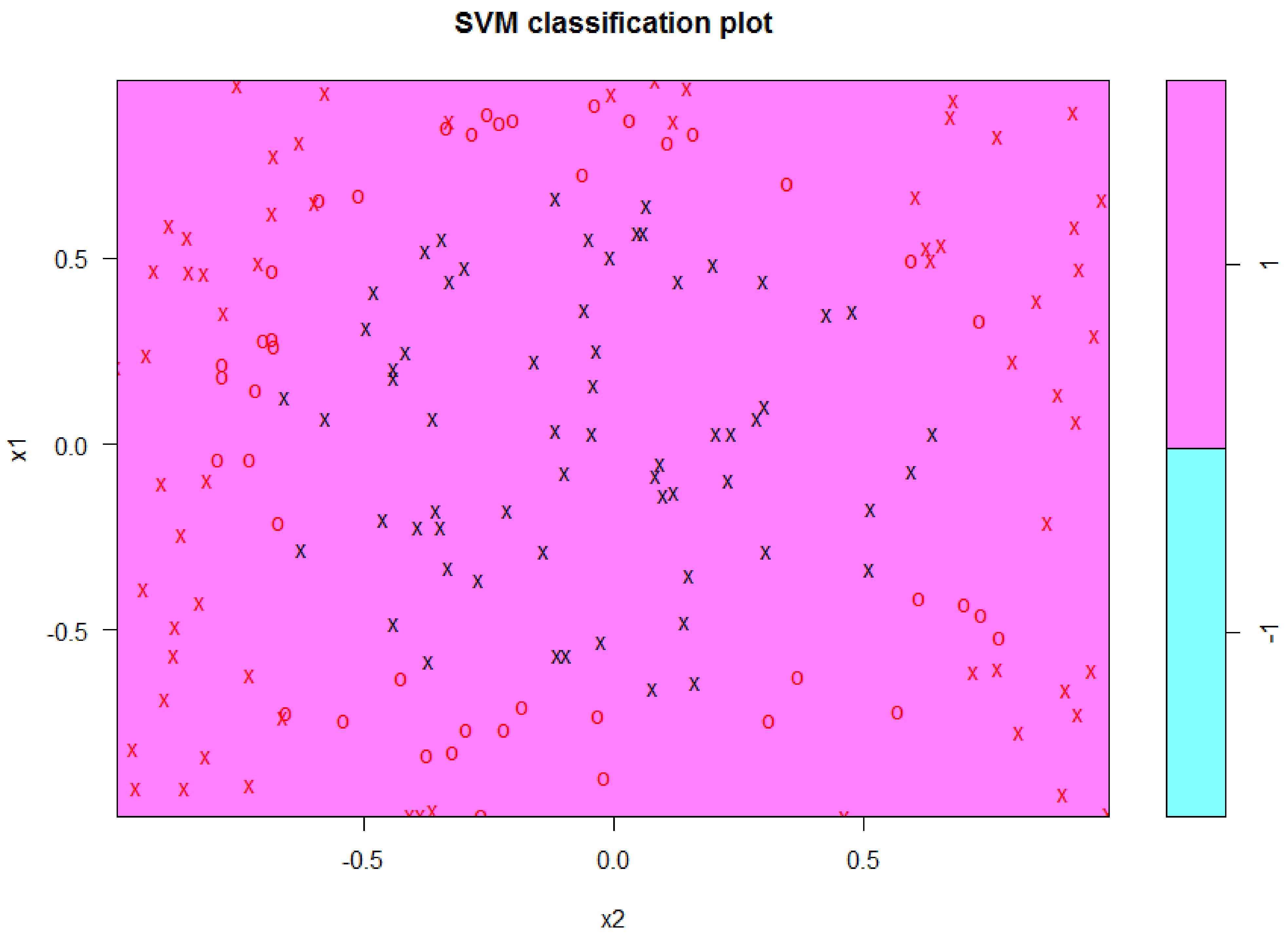
```
# Build default cost linear SVM on training set
svm_model <- svm(y ~ ., data = trainset, type = "C-classification", kernel = "linear")
svm_model
```

```
Number of Support Vectors:  126
```

```
# Calculate accuracy on test set
pred_test <- predict(svm_model, testset)
mean(pred_test == testset$y)
```

```
0.6129032
```

```
plot(svm_model, trainset)
```



SVM classification plot
x1
x2
0.5
0.0
-0.5
-0.5
0.0
0.5
1
-1

Linear SVM, cost = 100

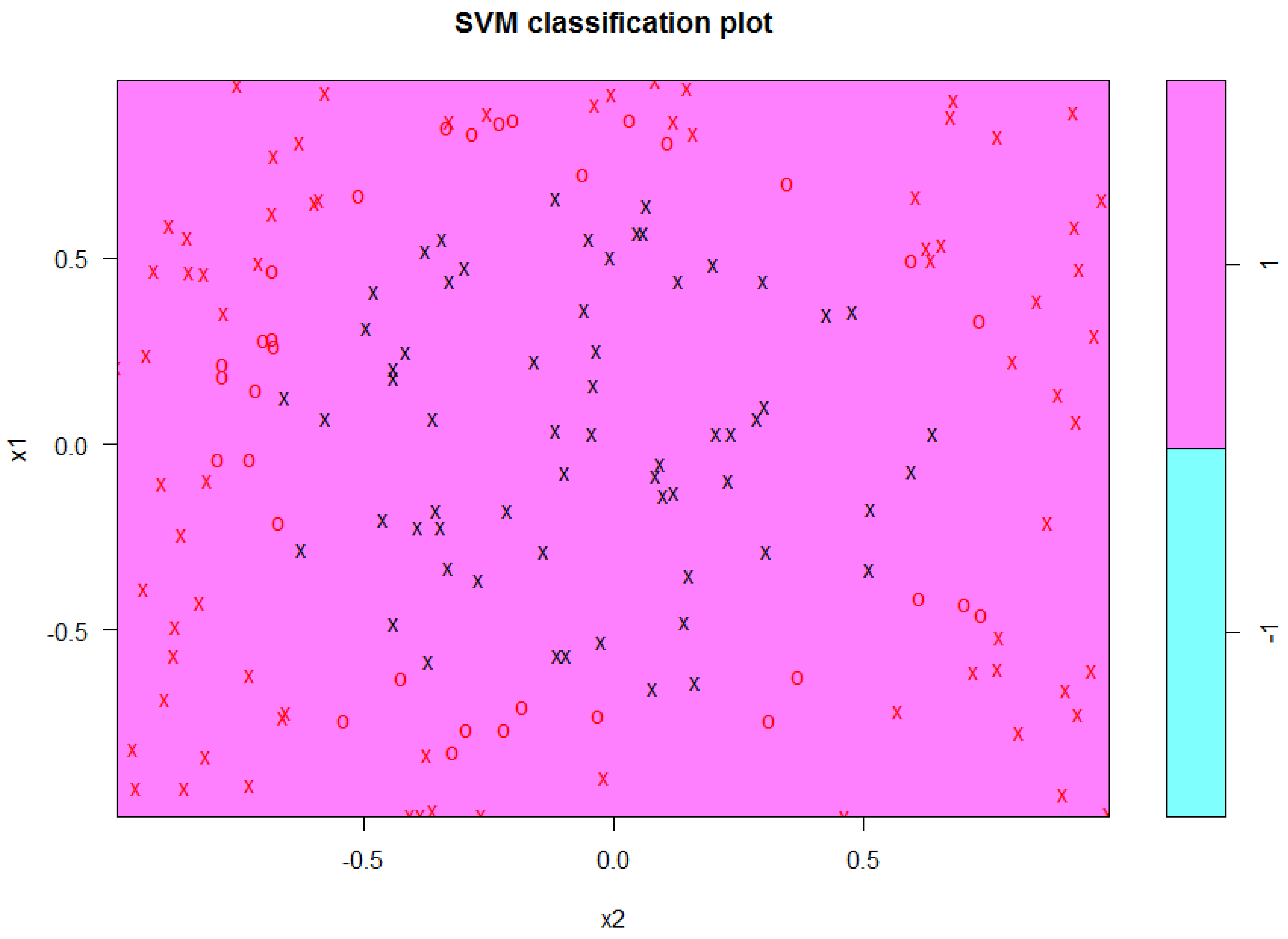
```r
svm_model <- svm(y ~ ., data = trainset, type = "C-classification", kernel = "linear")
svm_model
```

```
Number of Support Vectors:  136
```

```r
# Accuracy
pred_test <- predict(svm_model, testset)
mean(pred_test == testset$y)
```

```
0.6129032
```

```r
plot(svm_model, trainset)
```


SVM classification plot
x1
0.5
0.0
-0.5
x2
-0.5
0.0
0.5
1
-1

A better estimate of accuracy

- Calculate average accuracy over a number of independent train/test splits.
- Check standard deviation of result to get an idea of variability.

Average accuracy for default cost SVM

```
accuracy <- rep(NA, 100)
set.seed(10)
for (i in 1:100) {
  df[, "train"] <- ifelse(runif(nrow(df)) < 0.8, 1, 0)
  trainset <- df[df$train == 1, ]
  testset <- df[df$train == 0, ]
  trainColNum <- grep("train", names(trainset))
  trainset <- trainset[, -trainColNum]
  testset <- testset[, -trainColNum]
  svm_model<- svm(y ~ ., data = trainset, type = "C-classification", cost = 1, kernel = "linear")
  pred_test <- predict(svm_model, testset)
  accuracy[i] <- mean(pred_test == testset$y)}
mean(accuracy)
sd(accuracy)
```

```
0.642843
0.07606017
```

How well does a linear SVM perform?

- **Marginally better than a coin toss!**
- We can use our knowledge of the boundary to do much better.

Time to practice!

SUPPORT VECTOR MACHINES IN R

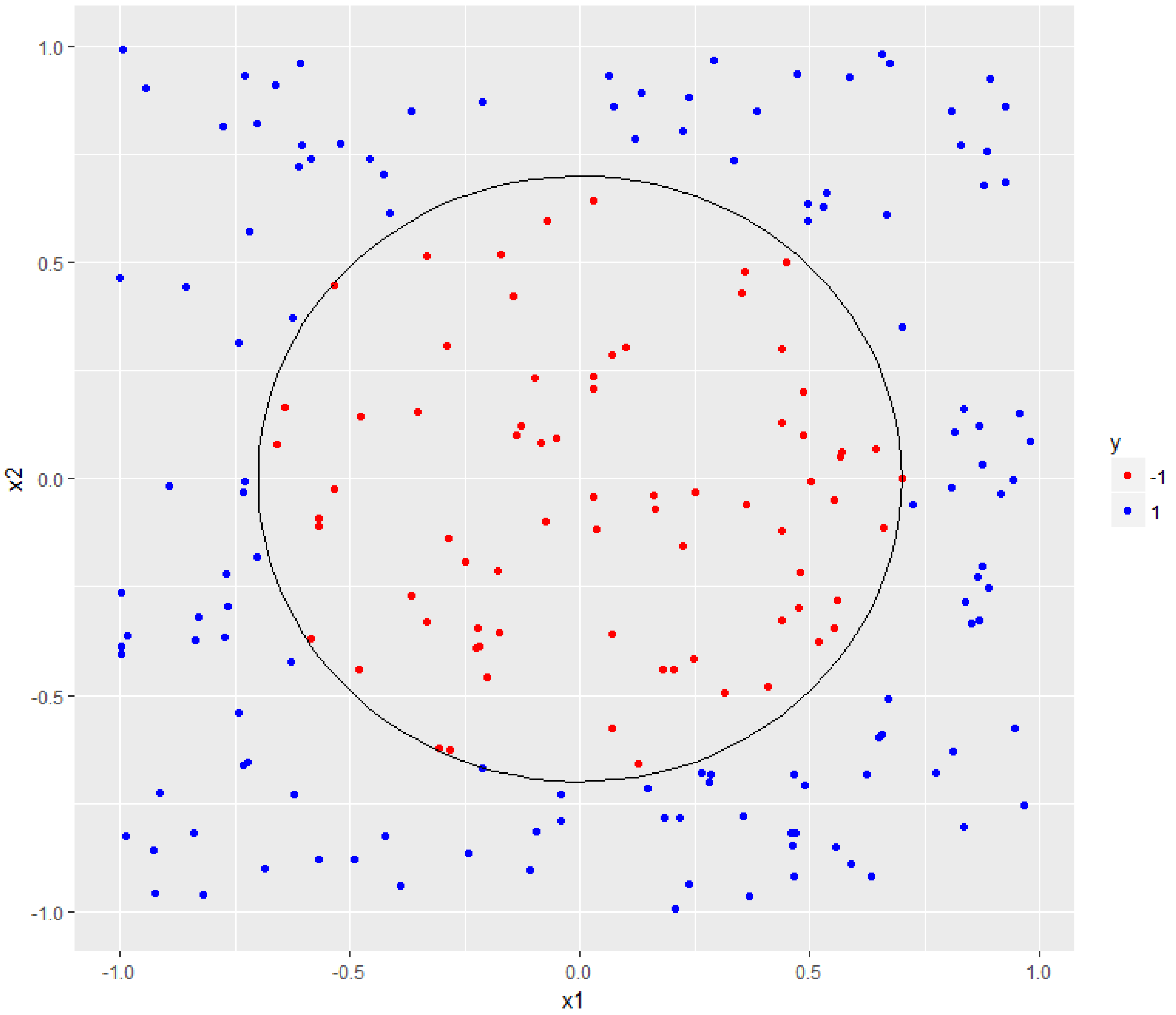
The kernel trick

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

The basic idea

- Devise a transformation that makes the problem linearly separable.
- We'll see how to do this for a radially separable dataset.



1.0
0.5
0.0
-0.5
-1.0
x2
-1.0
-0.5
0.0
0.5
1.0
x1
y
-1
1

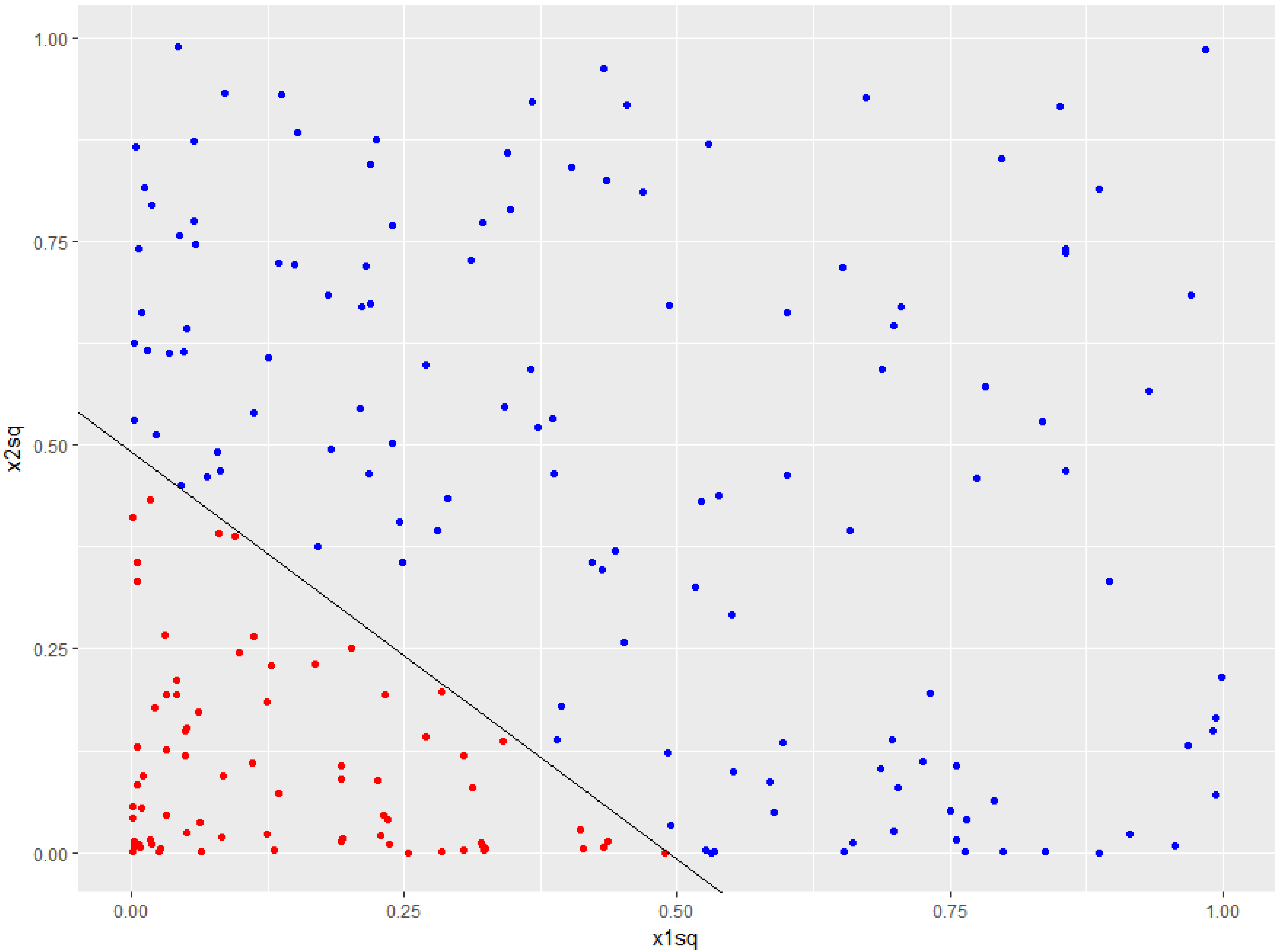
Transforming the problem

- Equation of boundary is $x_1^2 + x_2^2 = 0.49$
- Map x_1^2 to a new variable X_1 and x_2^2 to X_2
- The equation of boundary in the $X_1 - X_2$ space becomes...
- $X_1 + X_2 = 0.49$ (a line!!)

Plot in X1-X2 space - code

- Use `ggplot()` to plot the dataset in $X_1 - X_2$ space
- Equation of boundary $X_2 = -X_1 + 0.49$:
  - $slope = -1$
  - $yintercept = 0.49$

```
p <- ggplot(data = df4, aes(x = x1sq, y = x2sq, color = y)) +
  geom_point() +
  scale_color_manual(values = c("red", "blue")) +
  geom_abline(slope = -1, intercept = 0.49)

p
```


1.00
0.75
0.50
0.25
0.00
x2sq
0.00
0.25
0.50
0.75
1.00
x1sq

The Polynomial Kernel - Part 1

- Polynomial kernel: `(gamma * (u.v) + coef0) ^ degree`
  - `degree` is the degree of the polynomial
  - `gamma` and `coef0` are tuning parameters
  - `u`, `v` are vectors (datapoints) belonging to the dataset
- We can guess we need a 2nd degree polynomial (transformation)

Kernel functions

- The math formulation of SVMs requires transformations with specific properties.
- Functions satisfying these properties are called **kernel functions**
- Kernel functions are generalizations of vector dot products
- *Basic idea** - use a kernel that separates the data well!

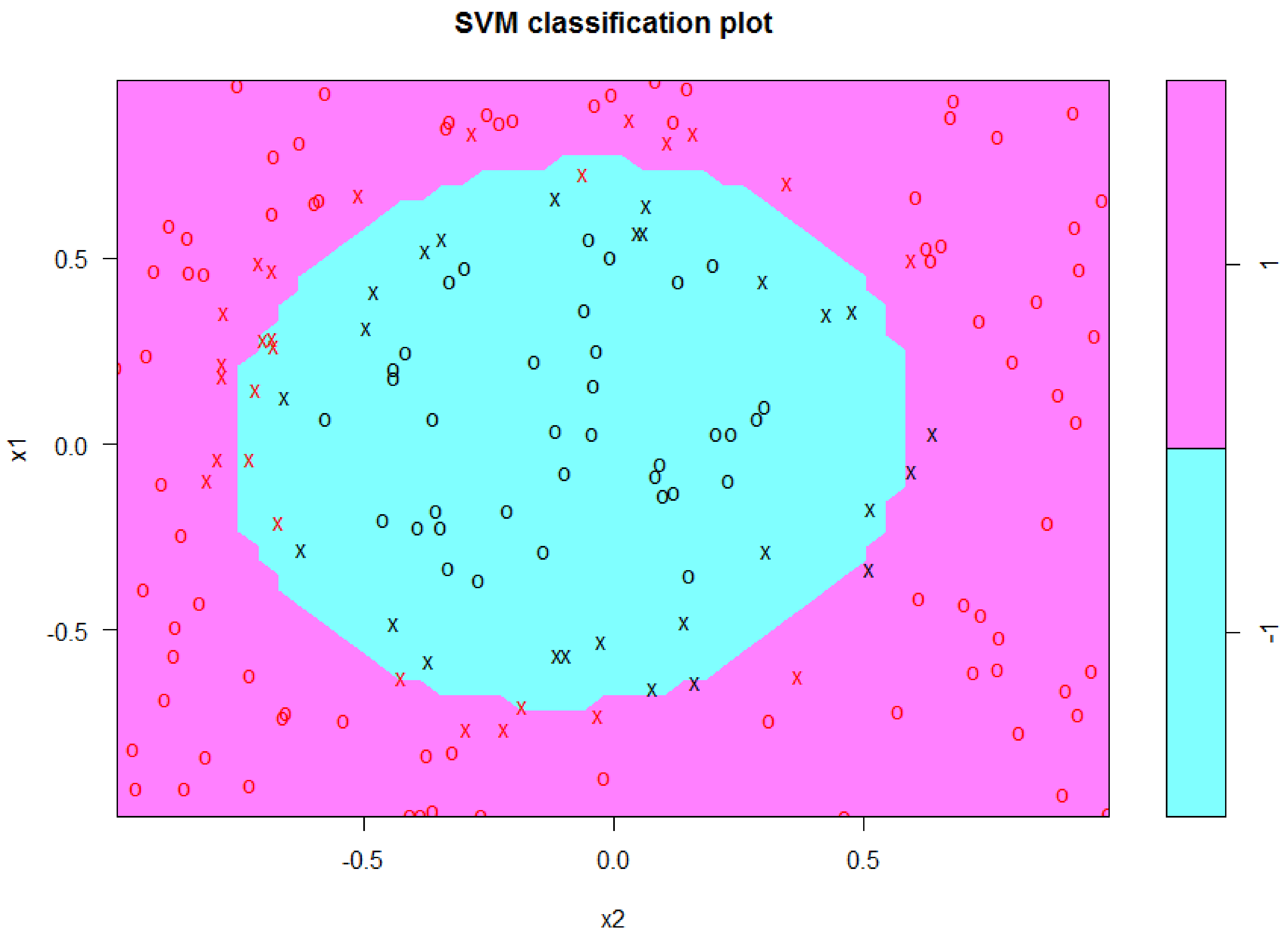
Radially separable dataset - quadratic kernel

- 80/20 train/test split
- Build a quadratic SVM for the radially separable dataset:
  - Set `degree = 2`
  - Set default values of `cost` , `gamma` and `coef0` (1, 1/2 and 0)

```
svm_model <- svm(y ~ ., data = trainset, type = "C-classification", kernel = "polynomial", degree = 2)
# Predictions
pred_test <- predict(svm_model, testset)
mean(pred_test == testset$y)
```

```
0.9354839
```

```
# Visualize model
plot(svm_model, trainset)
```


SVM classification plot
x1
0.5
0.0
-0.5
x2
-0.5
0.0
0.5
1
-1

Time to practice!

SUPPORT VECTOR MACHINES IN R

Tuning SVMs

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

Objective of tuning

- Hard to find optimal values of parameters manually for complex kernels.
- **Objective:** to find optimal set of parameters using `tune.svm()` function.

Tuning in a nutshell

- How it works:
  - set a range of search values for each parameter. Examples: `cost = 10^(-1:3)` , `gamma = c(0.1,1,10)` , `coef0 = c(0.1,1,10)`
  - Build an SVM model for each possible combination of parameter values and evaluate accuracy.
  - Return the parameter combination that yields the best accuracy.
- Computationally intensive procedure!

- Tune SVM model for the radially separable dataset created earlier
  - Built polynomial kernel SVM in previous lesson
  - Accuracy of SVM was ~94%.
- Can we do better by tuning gamma, cost and coef0?

```
tune_out <- tune.svm(x = trainset[,-3], y = trainset[,3],
                     type = "C-classification", kernel = "polynomial", degree = 2,
                     cost = 10^(-1:2), gamma = c(0.1,1,10), coef0 = c(0.1,1,10))
#print out tuned parameters
tune_out$best.parameters$cost
tune_out$best.parameters$gamma
tune_out$best.parameters$coef0
```

```
0.1
10
1
```

- Build SVM model using best values of parameters from `tune.svm()` .

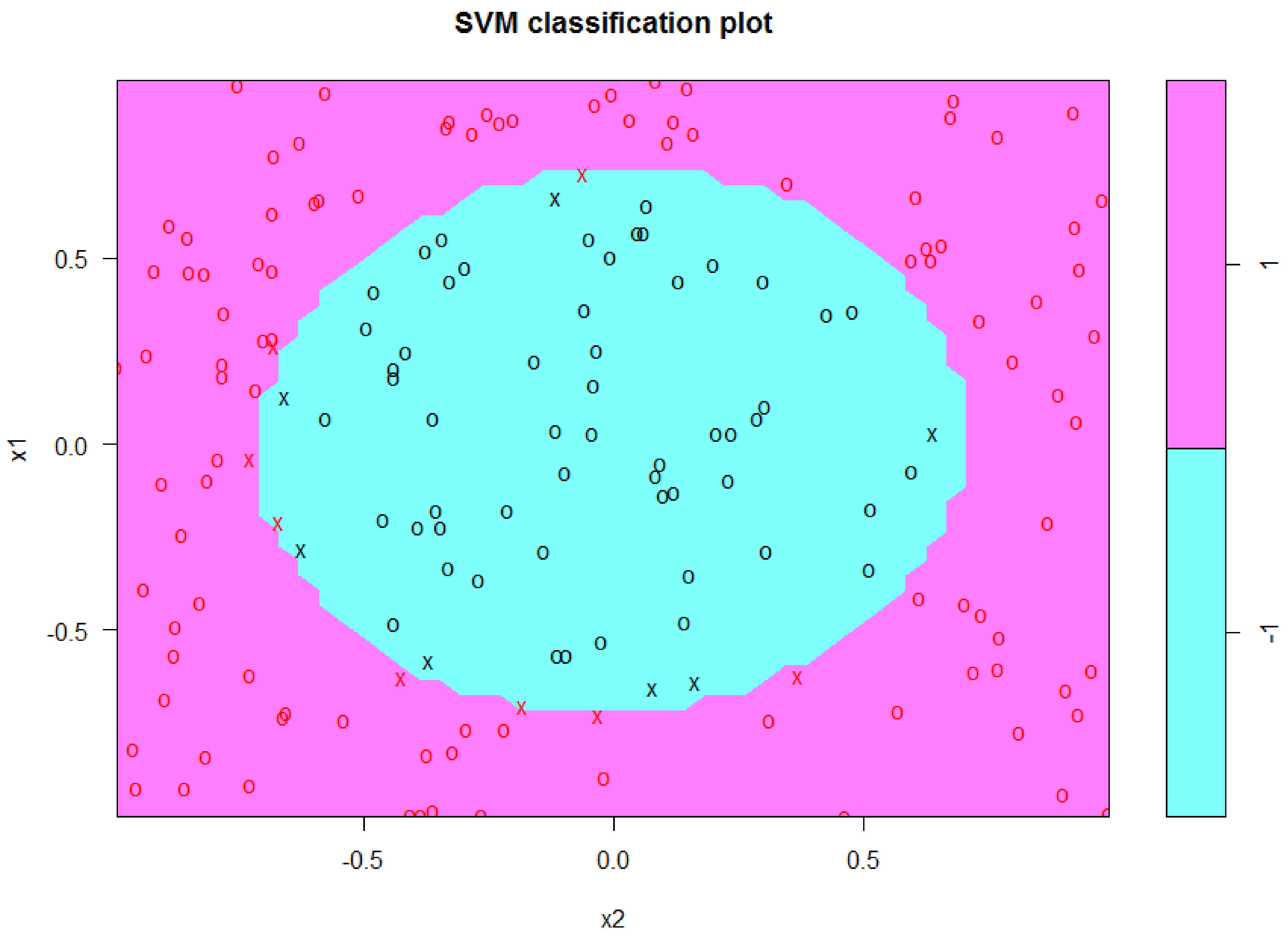
```
svm_model <- svm(y ~ ., data = trainset, type = "C-classification", kernel = "polynomial", degree = 2,
        cost = tune_out$best.parameters$cost,
        gamma = tune_out$best.parameters$gamma,
        coef0 = tune_out$best.parameters$coef0)
```

- evaluate training and test accuracy

```
pred_train <- predict(svm_model, trainset)
mean(pred_train == trainset$y)
pred_test <- predict(svm_model, testset)
mean(pred_test == testset$y)
```

```
1
0.9677419
```

```
#plot using svm plot
plot(svm_model, trainset)
```


SVM classification plot
x1
0.5
0.0
-0.5
x2
-0.5
0.0
0.5
1
-1

Time to practice!

SUPPORT VECTOR MACHINES IN R