Introduction

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

Preliminaries

- **Objective**: gain understanding of how SVMs work; options available in the algorithm and situations in which they work best.
- **Prerequisites:** Intermediate knowledge of R; basic visualization using `ggplot()` .
- **Approach:** Start with 1-dimensional example and gradually move on to more complex examples.

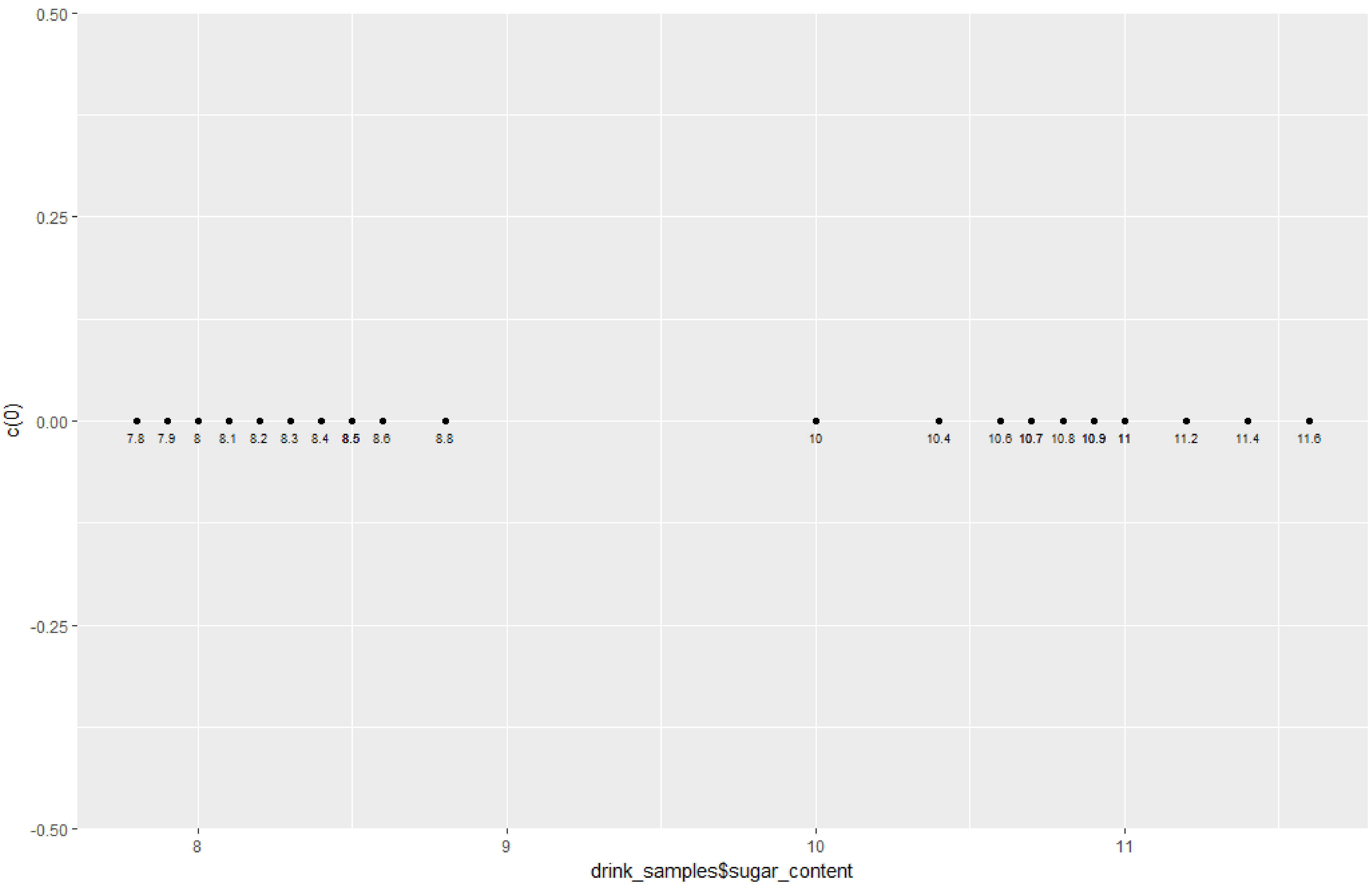
Sugar content of soft drinks

- Soft drink manufacturer has two versions of flagship brand:
  - **Choke** - sugar content 11g/ 100 ml.
  - **Choke-R** - sugar content 8 g/ 100 ml.
- Actual sugar content varies in practice.
- Given 25 samples chosen randomly, find a decision rule to determine brand.
- First step: visualize data!

Sugar content of soft drinks - visualization code

- Data in `drink_samples` dataframe.

```
# Specify dataframe, set plot aesthetics in geom_point (note y = 0)
p <- ggplot(drink_samples) +
  geom_point(aes(sugar_content, 0))
# Label each point with sugar content value, adjust text size and location
p <- p +
  geom_text(aes(sugar_content, 0, label = sugar_content),
            size = 2.5,
            vjust = 2,
            hjust = 0.5)
# Display plot
p
```


0.50
0.25
0.00
-0.25
-0.50
c(0)
7.8
7.9
8
8.1
8.2
8.3
8.4
8.5
8.6
8.8
10
10.4
10.6
10.7
10.8
10.9
11
11.2
11.4
11.6
8
9
10
11
drink_samples$sugar_content

Decision boundaries

- Let's pick two points in the interval as candidate boundaries:
  - 9.1 g/100 ml
  - 9.7 g/100 ml
- Classification (decision) rules:
  - if (y < 9.1) then "Choke-R" else "Choke"
  - if (y < 9.7) then "Choke-R" else "Choke"
- Let's visualize them on the plot shown on the previous slide.

Decision boundaries - visualization code

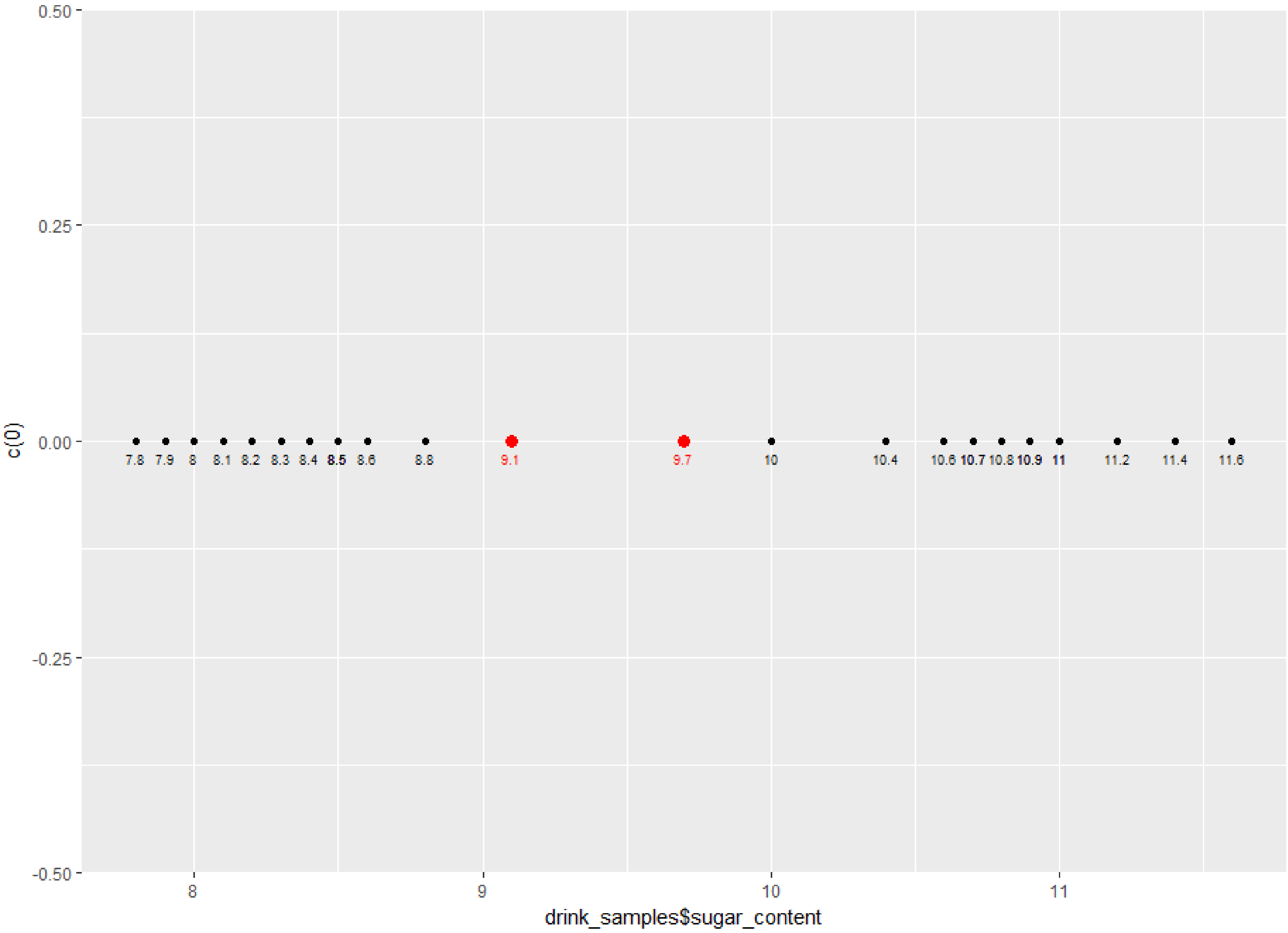
- Create a dataframe containing the two decision boundaries.

```
# Define data frame containing decision boundaries
d_bounds <- data.frame(sep = c(9.1, 9.7))
```

Decision boundaries - visualization code

- Add to plot using `geom_point()`

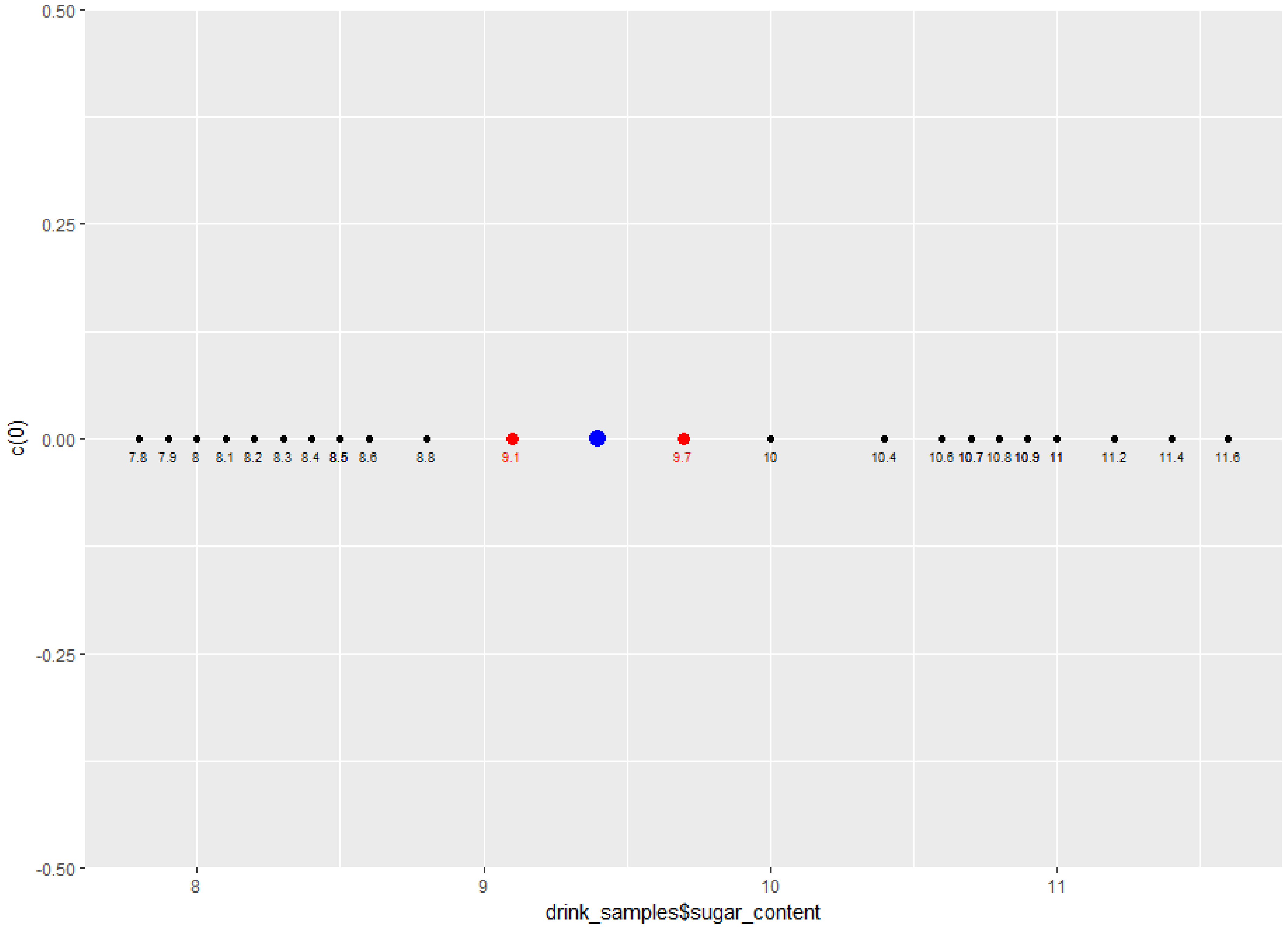
```
# Add decision boundaries to previous plot
p <- p +
  geom_point(data = d_bounds,
             aes(sep, 0),
             color = "red",
             size = 3) +
  geom_text(data = d_bounds,
            aes(sep, 0, label = sep),
            size = 2.5,
            vjust = 2,
            hjust = 0.5,
            color = "red")
# Display plot
p
```


0.50
0.25
0.00
-0.25
-0.50
c(0)
7.8 7.9 8 8.1 8.2 8.3 8.4 8.5 8.6 8.8 9.1 9.7 10 10.4 10.6 10.7 10.8 10.9 11 11.2 11.4 11.6
8
9
10
11
drink_samples$sugar_content

Maximum margin separator

- The best decision boundary is one that maximizes the margin: **maximal margin separator**
- Maximal margin separator lies halfway between the two clusters.
- Visualize the maximal margin separator.

```
# Create data frame with maximal margin separator
mm_sep <- data.frame(sep = c((8.8 + 10) / 2))
# Add mm boundary to previous plot
p <- p +
  geom_point(data = mm_sep,
             aes(sep, 0),
             color = "blue",
             size = 4)
# Display plot
p
```



0.50
0.25
0.00
-0.25
-0.50
c(0)
7.8
7.9
8
8.1
8.2
8.3
8.4
8.5
8.6
8.8
9.1
9.7
10
10.4
10.6
10.7
10.8
10.9
11
11.2
11.4
11.6
8
9
10
11
drink_samples$sugar_content

Time to practice!

SUPPORT VECTOR MACHINES IN R

Generating a linearly separable dataset

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

Overview of lesson

- Create a dataset that we'll use to illustrate key principles of SVMs.
- Dataset has two variables and a linear decision boundary.

Generating a two-dimensional dataset using runif()

- Generate a two variable dataset with 200 points
- Variables `x1` and `x2` uniformly distributed in (0,1).

```
# Preliminaries...
# Set required number of data points
n <- 200
# Set seed to ensure reproducibility
set.seed(42)
# Generate dataframe with two predictors x1 and x2 in (0,1)
df <- data.frame(x1 = runif(n),
                 x2 = runif(n))
```

Creating two classes

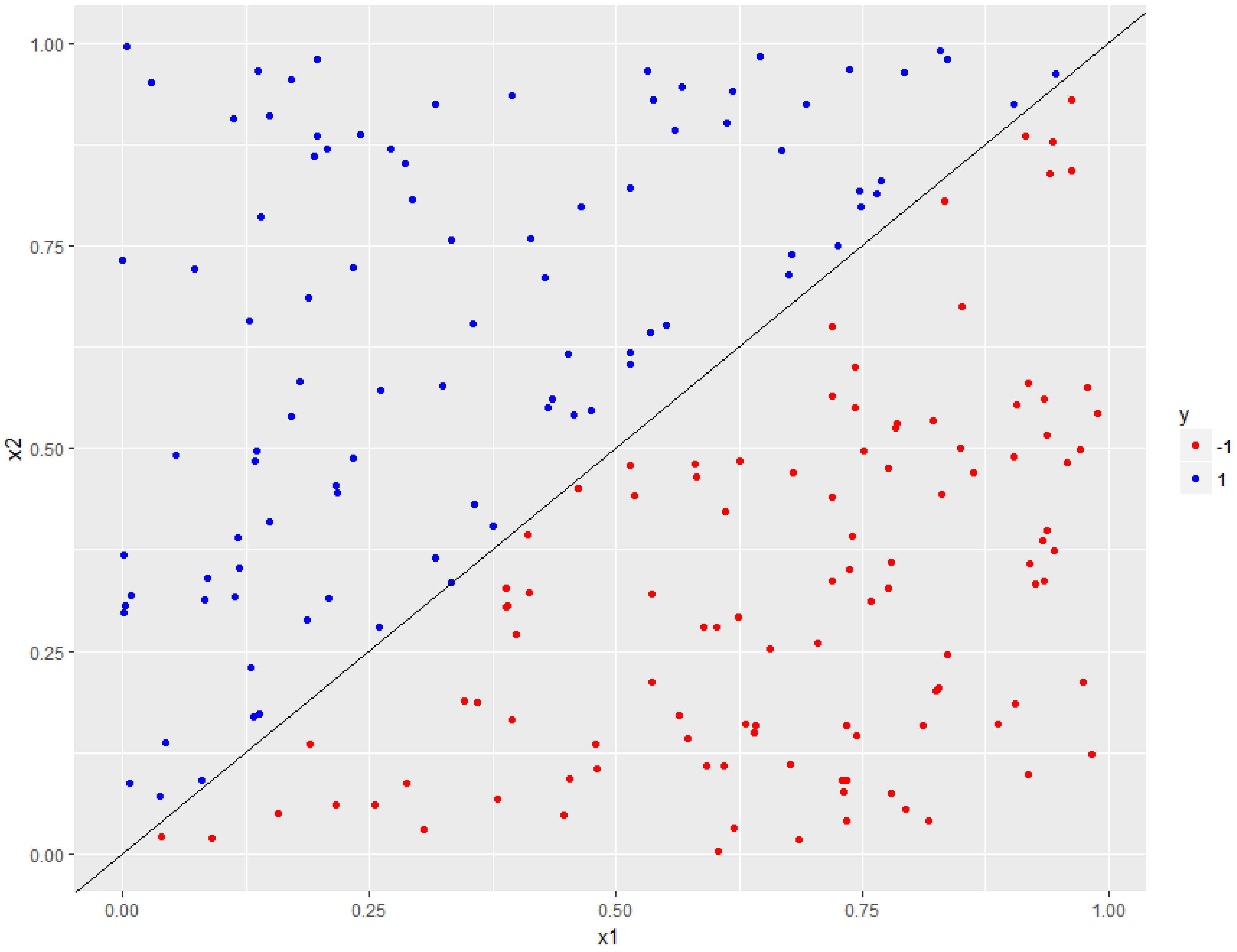
- Create two classes, separated by the straight line decision boundary `x1 = x2`
- Line passes through (0, 0) and makes a 45 degree angle with horizontal
- Class variable `y = -1` for points below line and `y = 1` for points above it

```
# Classify points as -1 or +1
df$y <- factor(ifelse(df$x1 - df$x2 > 0, -1, 1),
               levels = c(-1, 1))
```

Visualizing dataset using ggplot

- Create 2 dimensional scatter plot with x1 on the x axis and x2 on the y-axis
- Distinguish classes by color (below line = red; above line = blue)
- Decision boundary is line `x1 = x2` : passes through (0, 0) and has slope = 1

```
library(ggplot2)

# Build plot
p <- ggplot(data = df, aes(x = x1, y = x2, color = y)) +
     geom_point() +
     scale_color_manual(values = c("-1" = "red", "1" = "blue")) +
     geom_abline(slope = 1, intercept = 0)

# Display it
p
```

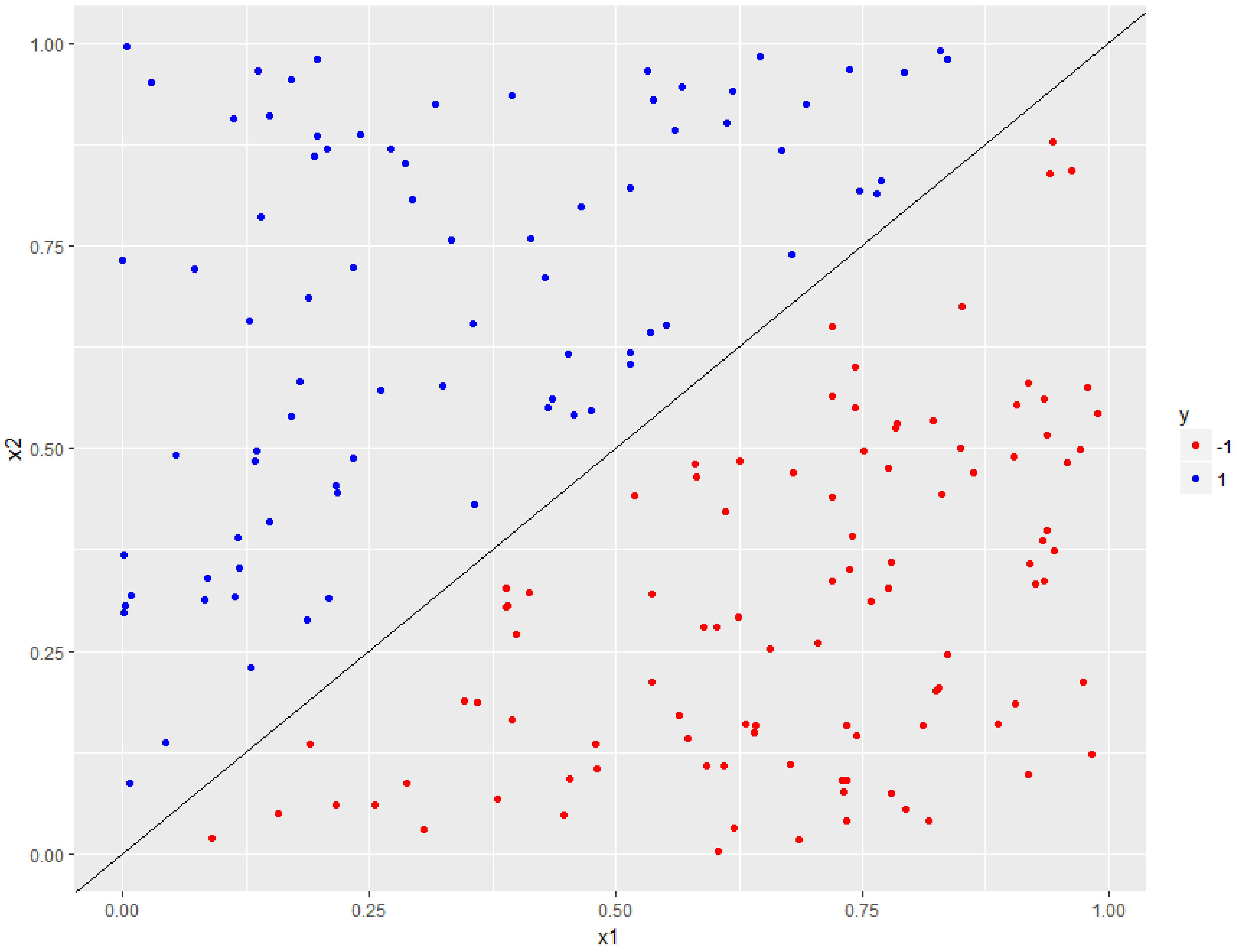


1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
-1
1

Introducing a margin

- To create a margin we need to remove points that lie close to the boundary
- Remove points that have x1 and x2 values that differ by less than a specified value

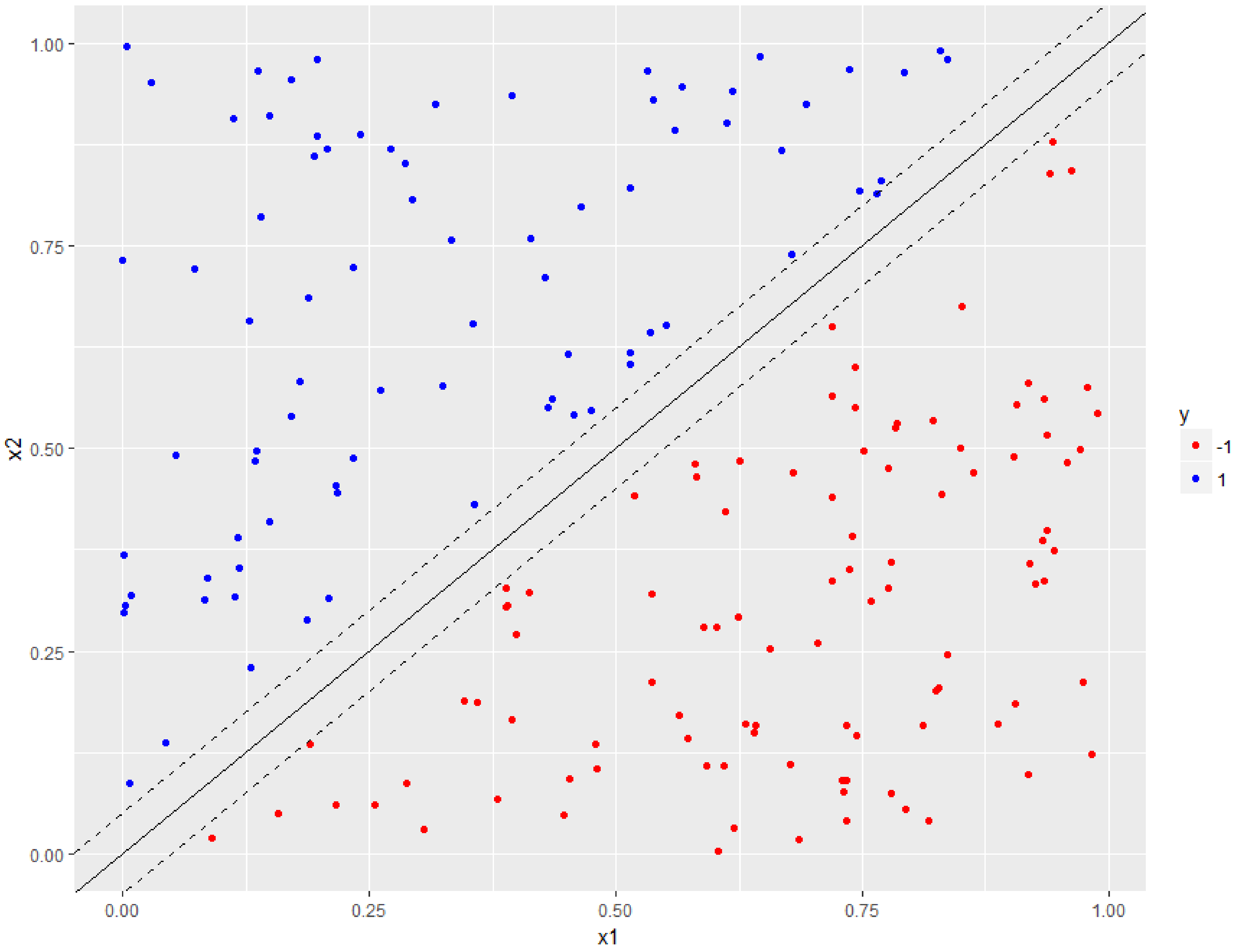
```
# Create a margin of 0.05 in dataset
delta <- 0.05
# Retain only those points that lie outside the margin
df1 <- df[abs(df$x1 - df$x2) > delta, ]
# Check number of data points remaining
nrow(df1)
# Replot dataset with margin (code is exactly same as before)
p <- ggplot(data = df1, aes(x = x1, y = x2, color = y)) +
     geom_point() +
     scale_color_manual(values = c("red", "blue")) +
     geom_abline(slope = 1, intercept = 0)
# Display plot
p
```


1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
-1
1

Plotting the margin boundaries

- The margin boundaries are:
  - parallel to the decision boundary (slope = 1).
  - located delta units on either side of it (delta = 0.05).

```
p <- p +
     geom_abline(slope = 1, intercept = delta, linetype = "dashed") +
     geom_abline(slope = 1, intercept = -delta, linetype = "dashed")


p
```



1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
-1
1

Time to practice!

SUPPORT VECTOR MACHINES IN R

Linear Support Vector Machines

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

Split into training and test sets

- The dataset generated in previous chapter is in dataframe `df` .
- Split dataset into training and test sets
- Random 80/20 split

```
# Set seed for reproducibility
set.seed(1)
# Set the upper bound for the number of rows to be in the training set
sample_size <- floor(0.8 * nrow(df))
# Assign rows to training/test sets randomly in 80/20 proportion
train <- sample(seq_len(nrow(df)), size = sample_size)
# Separate training and test sets
trainset <- df[train, ]
testset <- df[-train, ]
```

Decision boundaries and kernels

- Decision boundaries can have different shapes - lines, polynomials or more complex functions.
- Type of decision boundary is called a **kernel**.
- Kernel must be specified upfront.
- This chapter focuses on linear kernels.

SVM with linear kernel

- We'll use the svm function from the `e1071` library.
- The function has a number of parameters. We'll set the following explicitly:
  - **formula** - a formula specifying the dependent variable. y in our case.
  - **data** - dataframe containing the data - i.e. trainset.
  - **type** - set to C-classification (classification problem).
  - **kernel** - this is the form of the decision boundary, linear in this case.
  - **cost** and **gamma** - these are parameters that are used to tune the model.
  - **scale** - Boolean indicating whether to scale data.

Building a linear SVM

- Load e1071 library and invoke `svm()` function

```
library(e1071)
```

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

Overview of model

- Entering `svm_model` gives:
  - an overview of the model including classification and kernel type
  - tuning parameter values

```
svm_model
```

```
Call:
svm(formula = y ~ .,
    data = trainset,
    type = "C-classification",
    kernel = "linear",
    scale = FALSE)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5
Number of Support Vectors:  55
```

```r
# Index of support vectors in training dataset
svm_model$index
# Support vectors
svm_model$SV
# Negative intercept (unweighted)
svm_model$rho
# Weighting coefficients for support vectors
svm_model$coefs
```

```
4   8  10  11  18  37  38  39  47  59  60  74  76  77  78  80  83 ...
             x1          x2
5   0.519095949 0.44232464
-0.1087075
            [,1]
 [1,]  1.0000000
```

- Obtain class predictions for training and test sets.
- Evaluate the training and test set accuracy of the model.

```
# Training accuracy
pred_train <- predict(svm_model, trainset)
mean(pred_train == trainset$y)
```

```
1
```

```
# Test accuracy
pred_test <- predict(svm_model, testset)
mean(pred_test == testset$y)
```

```
1
# Perfect!!
```

Time to practice!

SUPPORT VECTOR MACHINES IN R

Visualizing linear SVMs

SUPPORT VECTOR MACHINES IN R

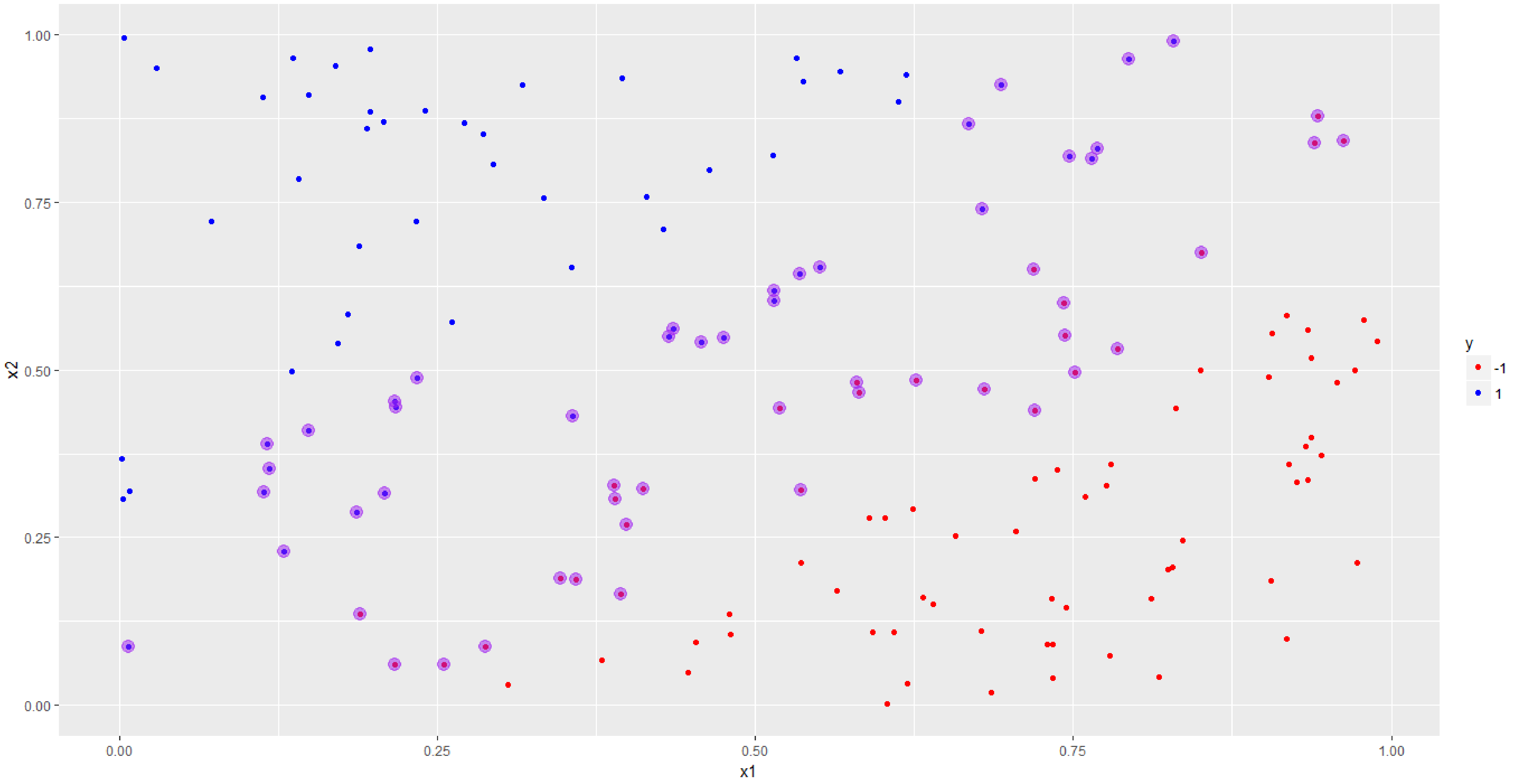
Kailash Awati
Instructor

- Plot the training data using `ggplot()`.

```
# Visualize training data, distinguish classes using color
p <- ggplot(data = trainset, aes(x = x1, y = x2, color = y)) +
     geom_point() +
     scale_color_manual(values = c("red", "blue"))
# Render plot
p
```

- Mark out the support vectors using `index` from `svm_model`.

```
# Identify support vectors
df_sv <- trainset[svm_model$index, ]
# Mark out support vectors in plot
p <- p + geom_point(data = df_sv,
                    aes(x = x1, y = x2),
                    color = "purple",
                    size = 4, alpha = 0.5)
# Display plot
p
```



1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
-1
1

Find slope and intercept of the boundary:

- Build the weight vector, `w`, from `coefs` and `SV` elements of `svm_model`.

```
# Build weight vector
w <- t(svm_model$coefs) %*% svm_model$SV
```

- slope = `-w[1] / w[2]`

```
# Calculate slope and save it to a variable
slope_1 <- -w[1] / w[2]
```

- intercept = `svm_model$rho / w[2]`

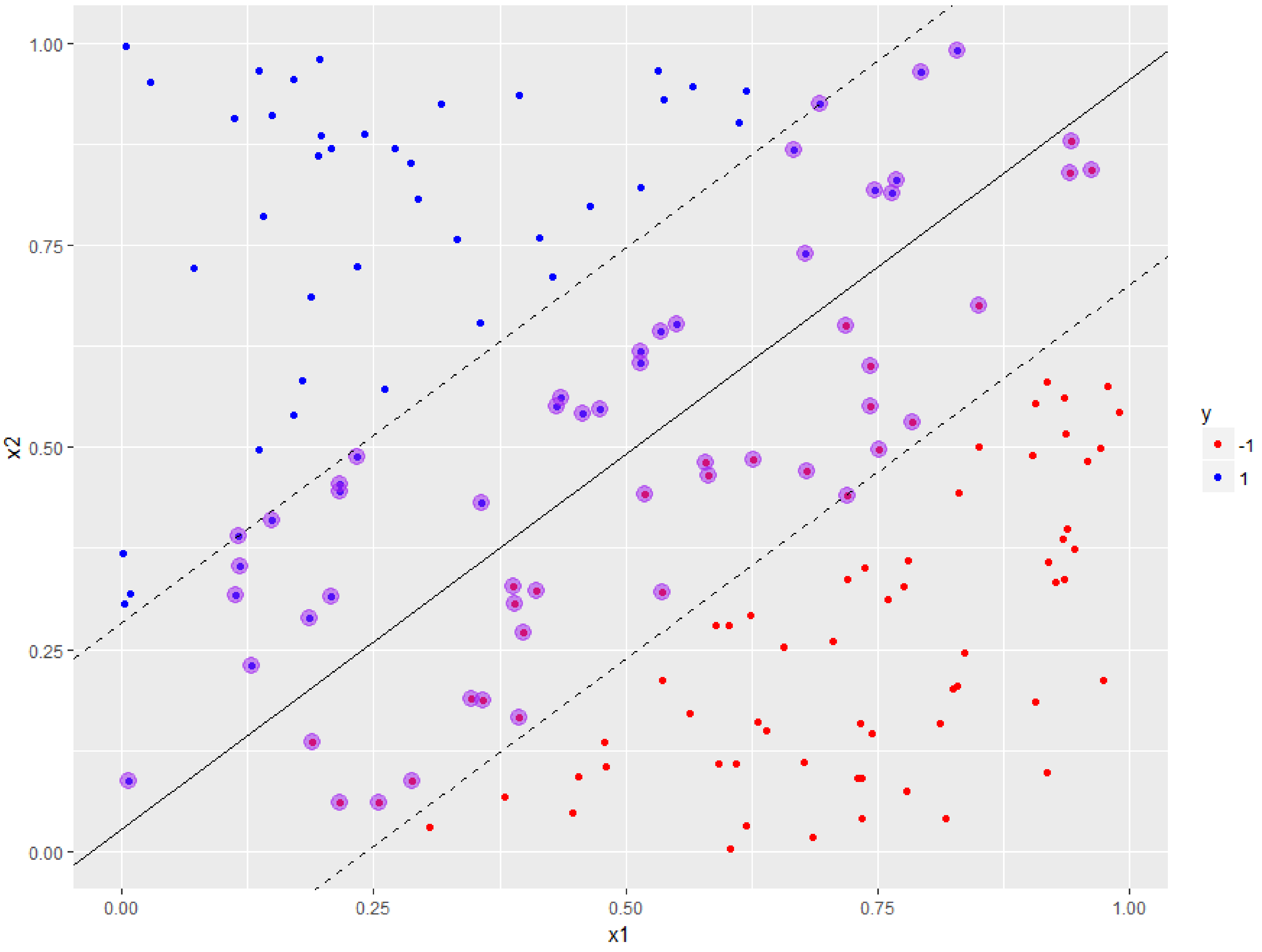
```
# Calculate intercept and save it to a variable
intercept_1 <- svm_model$rho / w[2]
```

- Add **decision boundary** using slope and intercept calculated in previous slide.
- We use `geom_abline()` to add the decision boundary to the plot.

```
# Plot decision boundary based on calculated slope and intercept
p <- p + geom_abline(slope = slope_1,
                     intercept = intercept_1)
```

- **Margins** parallel to decision boundary, offset by `1 / w[2]` on either side of it.

```
# Add margins to plot
p <- p +
    geom_abline(slope = slope_1,
                intercept = intercept_1 - 1 / w[2],
                linetype = "dashed") +
    geom_abline(slope = slope_1,
                intercept = intercept_1 + 1 / w[2],
                linetype = "dashed")
# Display plot
p
```



1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
-1
1

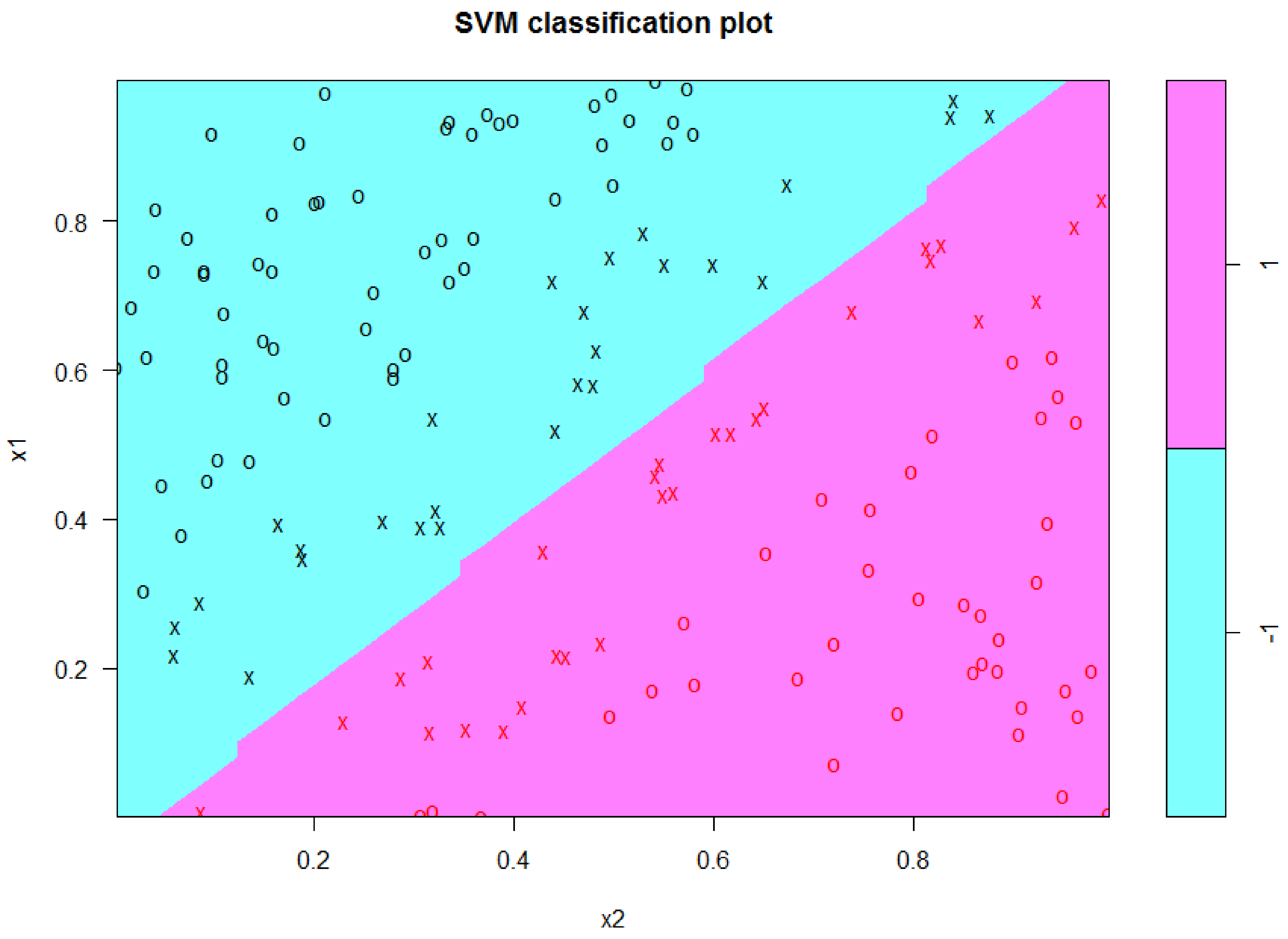
Soft margin classifiers

- Allow for uncertainty in location / shape of boundary
  - Never perfectly linear
  - Usually unknown
- Our decision boundary is linear, so we can reduce margin

Visualizing the decision boundary using the svm plot() function

- The svm `plot()` function in `e1071` offers an easy way to plot the decision boundary.

```
# Visualize decision boundary using built in plot function
plot(x = svm_model,
     data = trainset)
```


SVM classification plot
0.8
0.6
0.4
0.2
x1
0.2
0.4
0.6
0.8
x2
1
-1

Time to practice!

SUPPORT VECTOR MACHINES IN R

Tuning linear SVMs

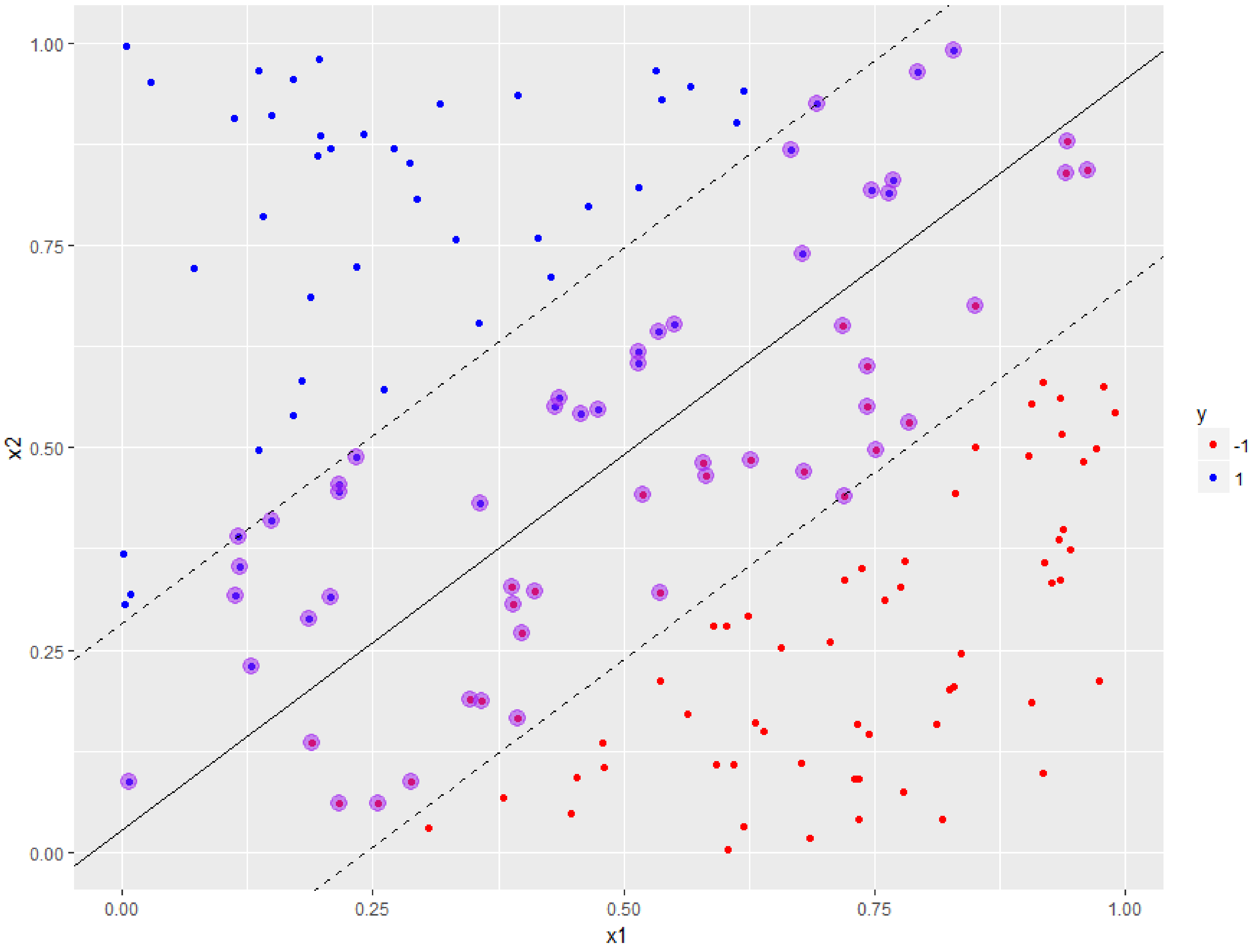
SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

Linear SVM, default cost

```r
library(e1071)
svm_model <- svm(y ~ .,
                data = trainset,
                type = "C-classification",
                kernel = "linear",
                scale = FALSE)
# Print model summary
svm_model
```

```
Call:
svm(formula = y ~ .,
    data = trainset,
    type = "C-classification",
    kernel = "linear",
    scale = FALSE)

Parameters:
SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5
Number of Support Vectors:  55
```

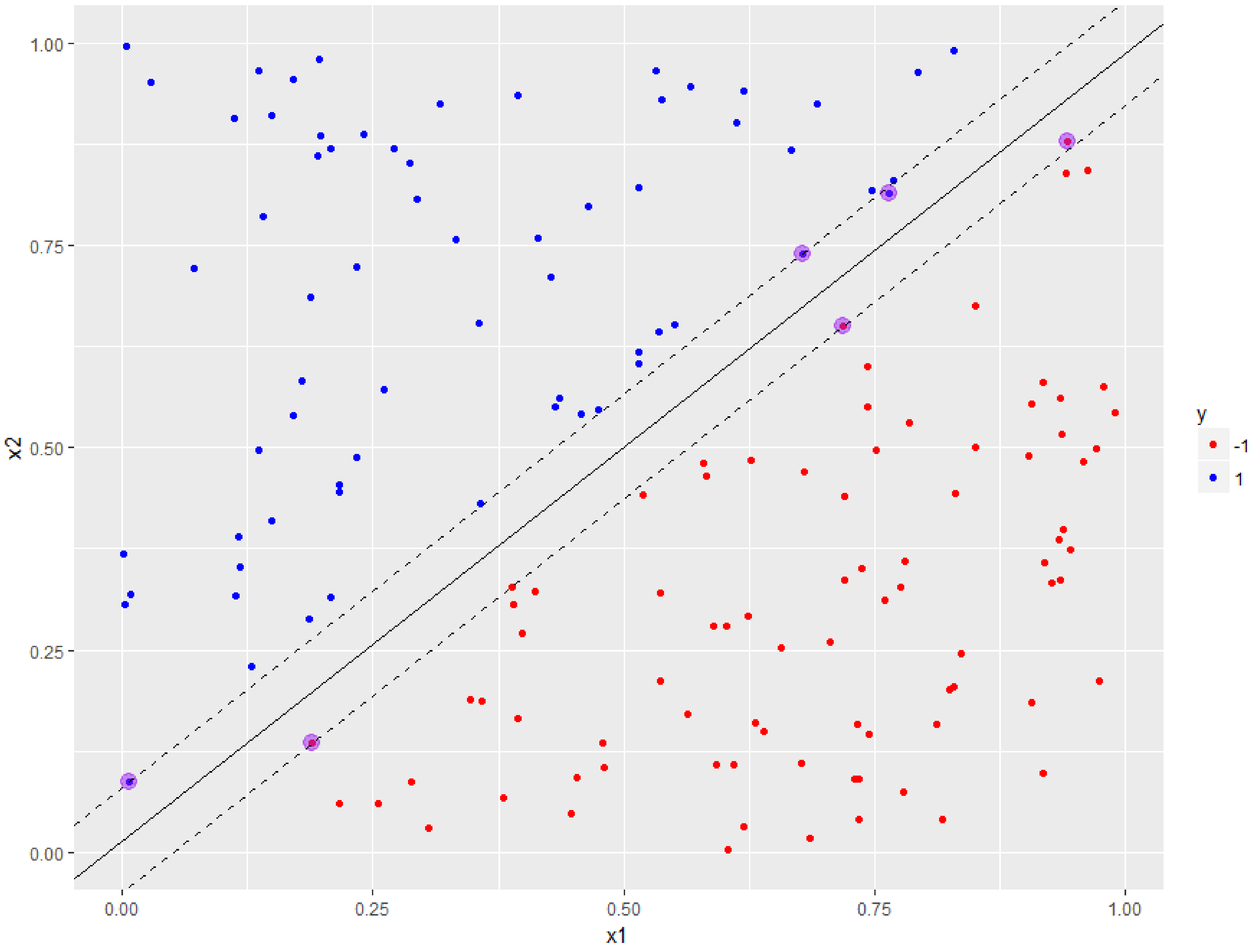


1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
-1
1

Linear SVM with cost = 100

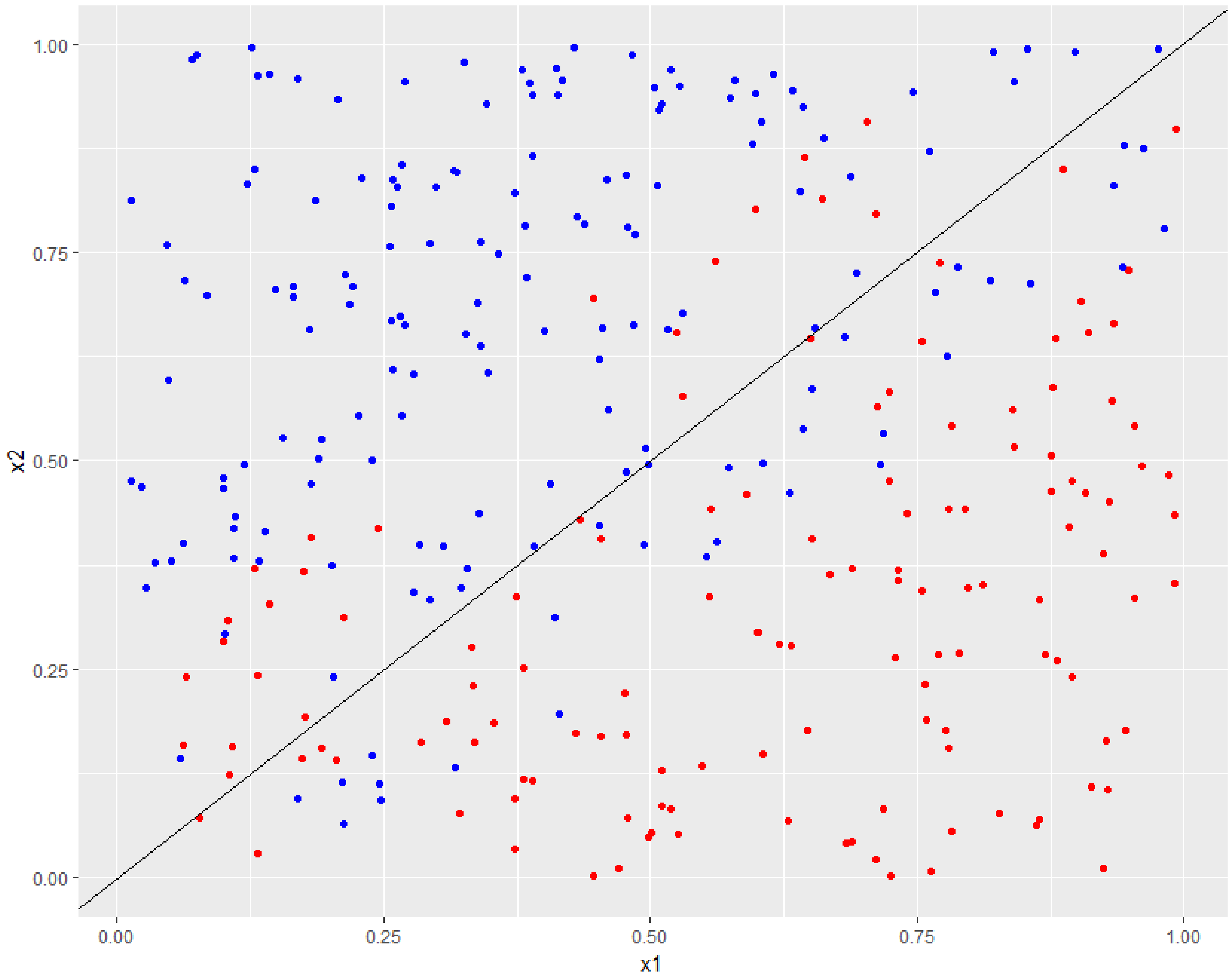
```r
library(e1071)
svm_model <- svm(y ~ .,
                data = trainset,
                type = "C-classification",
                kernel = "linear",
                cost = 100,
                scale = FALSE)
# Print model summary
svm_model
```

```
Call:
svm(formula = y ~ .,
    data = trainset,
    type = "C-classification",
    kernel = "linear",
    cost = 100,
    scale = FALSE)

Parameters:
SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  100
      gamma:  0.5
Number of Support Vectors:  6
```


1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
-1
1

Implication

- Can be useful to reduce margin if decision boundary is known to be linear
- ...but this is rarely the case in real life



1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1

Nonlinear dataset, linear SVM (cost = 100)

- Build cost=100 model using training set composed of 80% of data

```
# Build model
library(e1071)
svm_model<- svm(y ~ .,
                data = trainset,
                type = "C-classification",
                kernel = "linear",
                cost = 100,
                scale = FALSE)
```

- Calculate accuracy

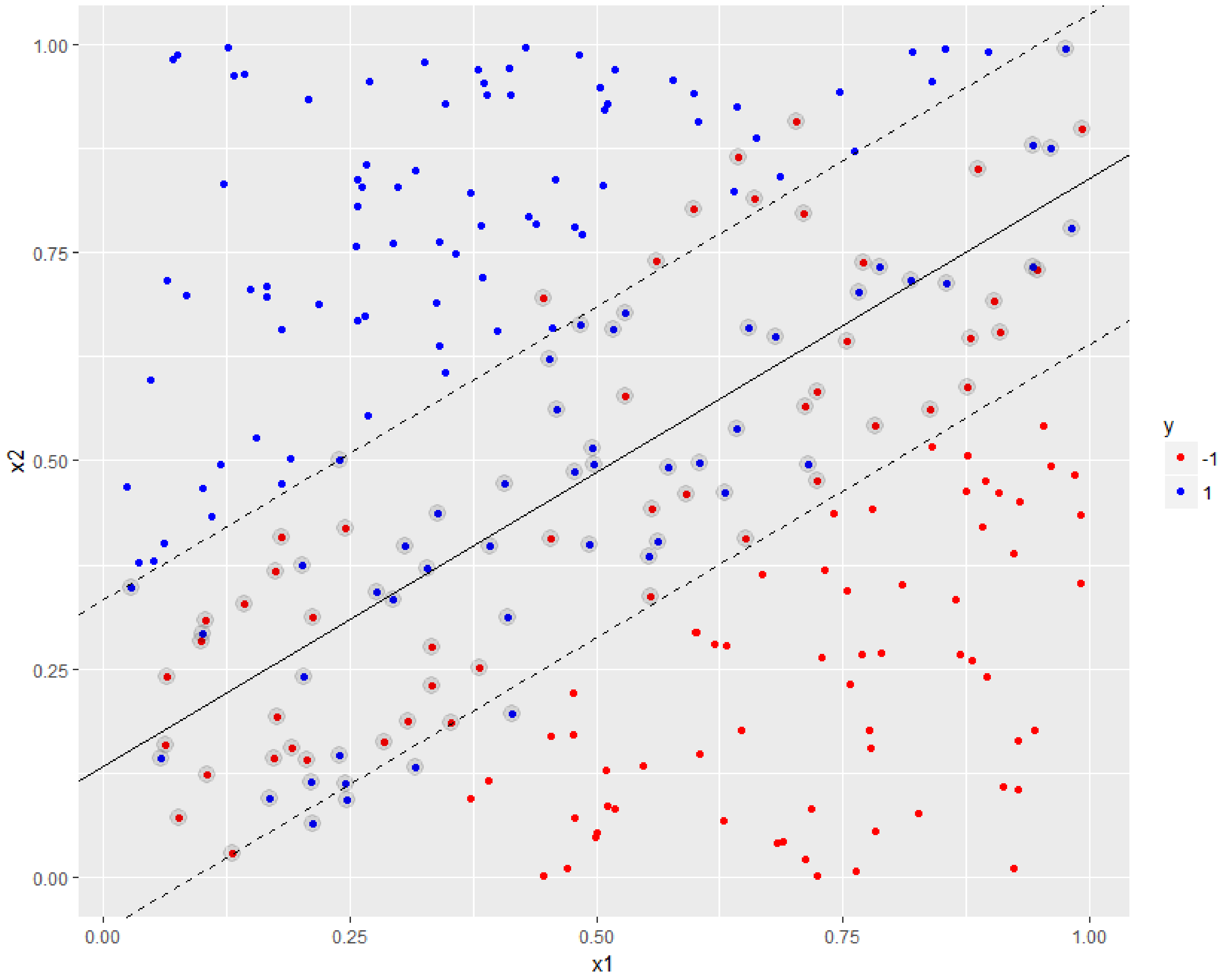
```
# Train and test accuracy
pred_train <- predict(svm_model, trainset)
mean(pred_train == trainset$y)
```

```
0.8208333
```

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

```
0.85
```

- Average test accuracy over 50 random train/test splits: 82.9%



1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
-1
1

Nonlinear dataset, linear SVM (cost = 1)

- Rebuild model setting cost =1

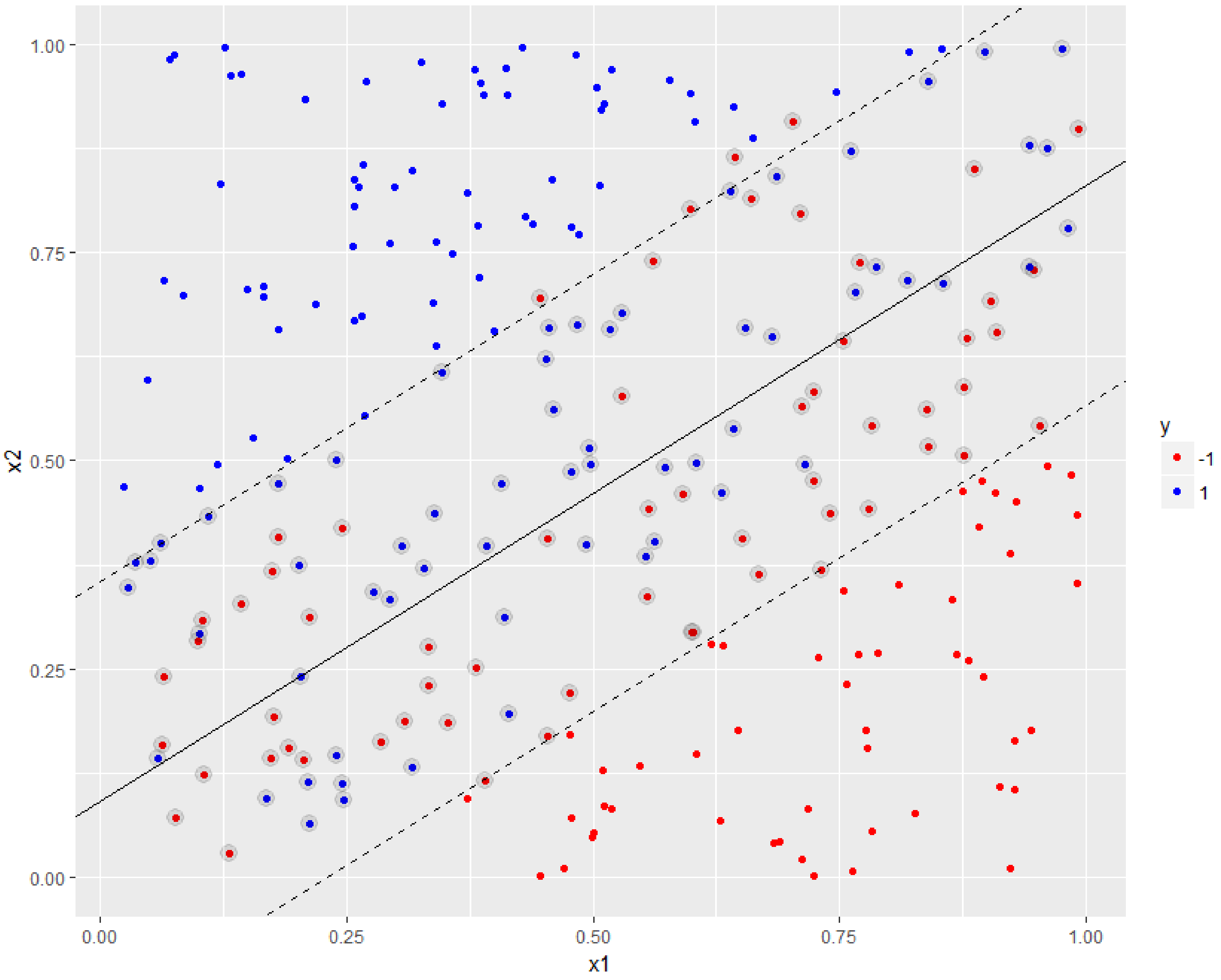
```
# Trainset contains 80% of data
# Same train/test split as before.
# Build model
svm_model <- svm(y ~ .,
                data = trainset,
                type = "C-classification",
                kernel = "linear",
                cost = 1,
                scale = FALSE)
```

- Calculate test accuracy

```
# Test accuracy
pred_test <- predict(svm_model, testset)
mean(pred_test == testset$y)
```

```
0.8666667
```

- Average test accuracy over 50 random train/test splits: 83.7%



1.00
0.75
0.50
0.25
0.00
x2
0.00
0.25
0.50
0.75
1.00
x1
y
-1
1

Time to practice!

SUPPORT VECTOR MACHINES IN R

Multiclass problems

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

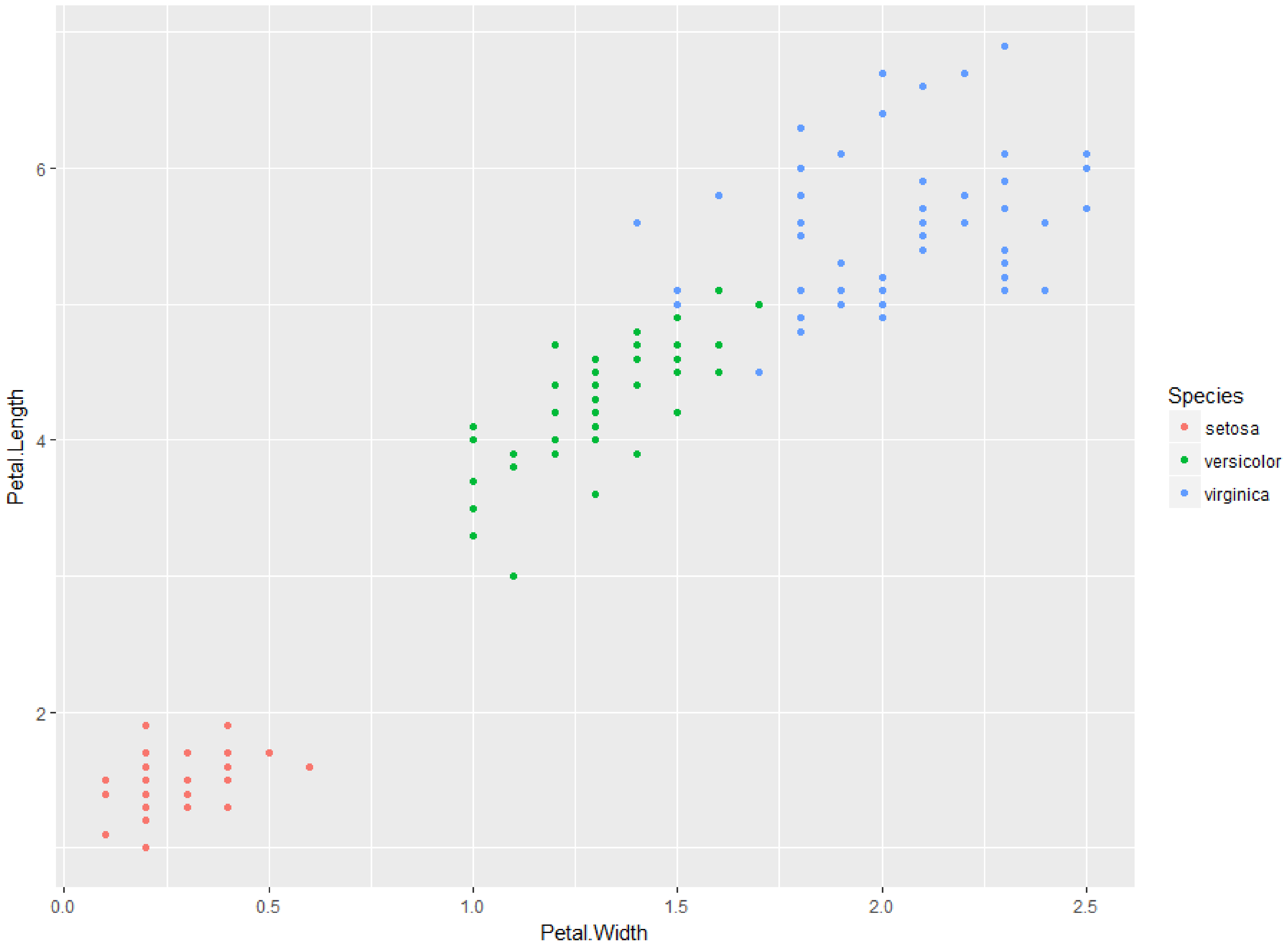
The iris dataset - an introduction

- 150 measurements of 5 attributes
  - Petal width and length - number (predictor variables)
  - Sepal width and length - number (predictor variables)
  - Species - category: setosa, virginica or versicolor (predicted variable)
- Dataset available from **UCI ML repository**

Visualizing the iris dataset

- Plot petal length vs petal width.

```
library(ggplot2)

# Plot petal length vs width for dataset, distinguish species by color
p <- ggplot(data = iris,
            aes(x = Petal.Width,
                y = Petal.Length,
                color = Species)) +
     geom_point()

# Display plot
p
```



6
4
2
Petal.Length
0.0
0.5
1.0
1.5
2.0
2.5
Petal.Width
Species
setosa
versicolor
virginica

How does the SVM algorithm deal with multiclass problems?

- SVMs are essentially binary classifiers.
- Can be applied to multiclass problems using the following voting strategy:
  - Partition the data into subsets containing two classes each.
  - Solve the binary classification problem for each subset.
  - Use majority vote to assign a class to each data point.
- Called **one-against-one** classification strategy.

Building a multiclass linear SVM

- Build a linear SVM for the iris dataset
  - 80/20 training / test split (seed 10), default cost

```
library(e1071)

# Build model
svm_model <- svm(Species ~ .,
                 data = trainset,
                 type = "C-classification",
                 kernel = "linear")
```

- Calculate accuracy

```
pred_train <- predict(svm_model, trainset)
mean(pred_train == trainset$Species)
```

```
0.9756098
```

```
pred_test <- predict(svm_model, testset)
mean(pred_test == testset$Species)
```

```
0.962963
```

Time to practice!

SUPPORT VECTOR MACHINES IN R

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))
```

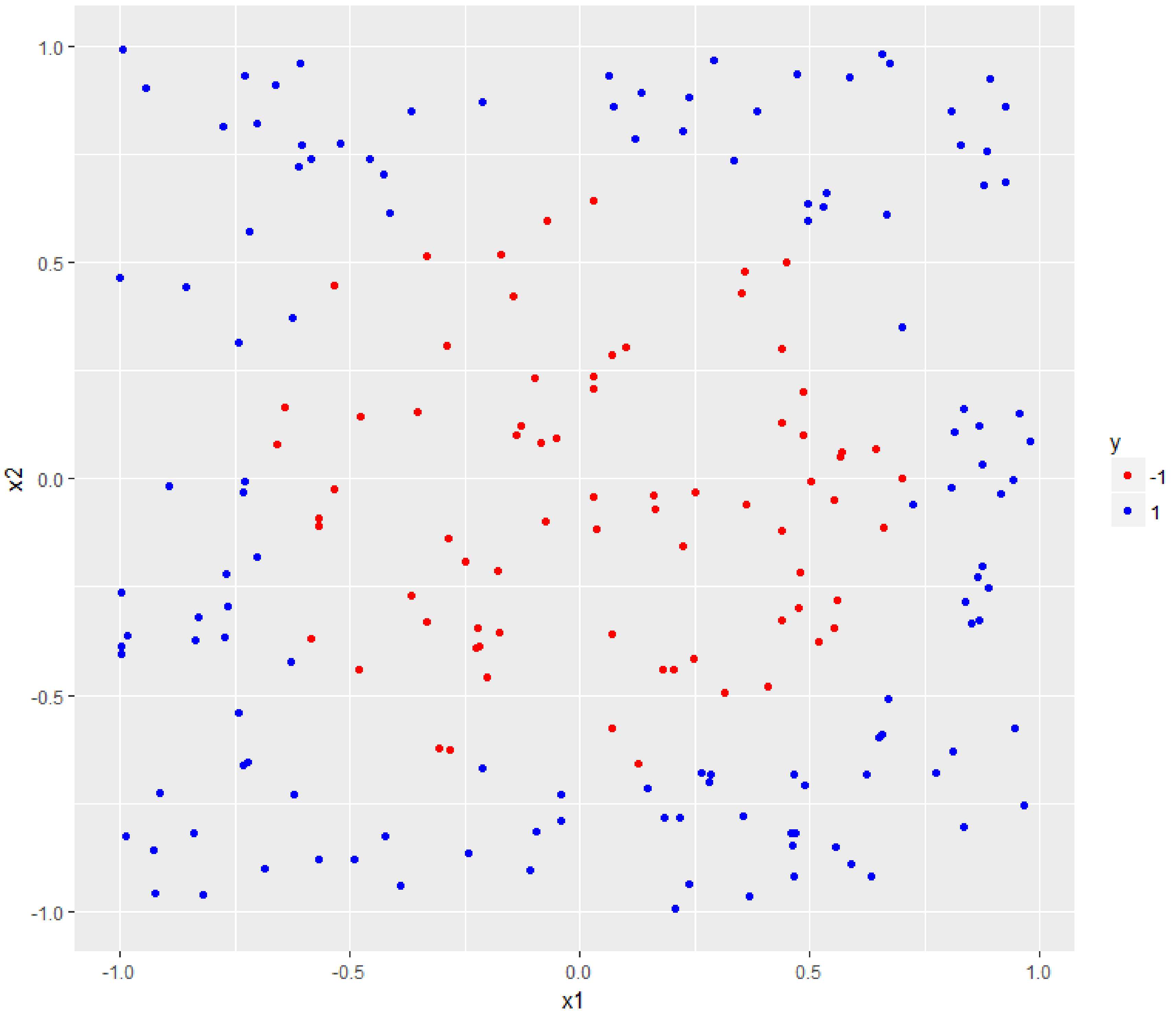
Plot the dataset

- 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

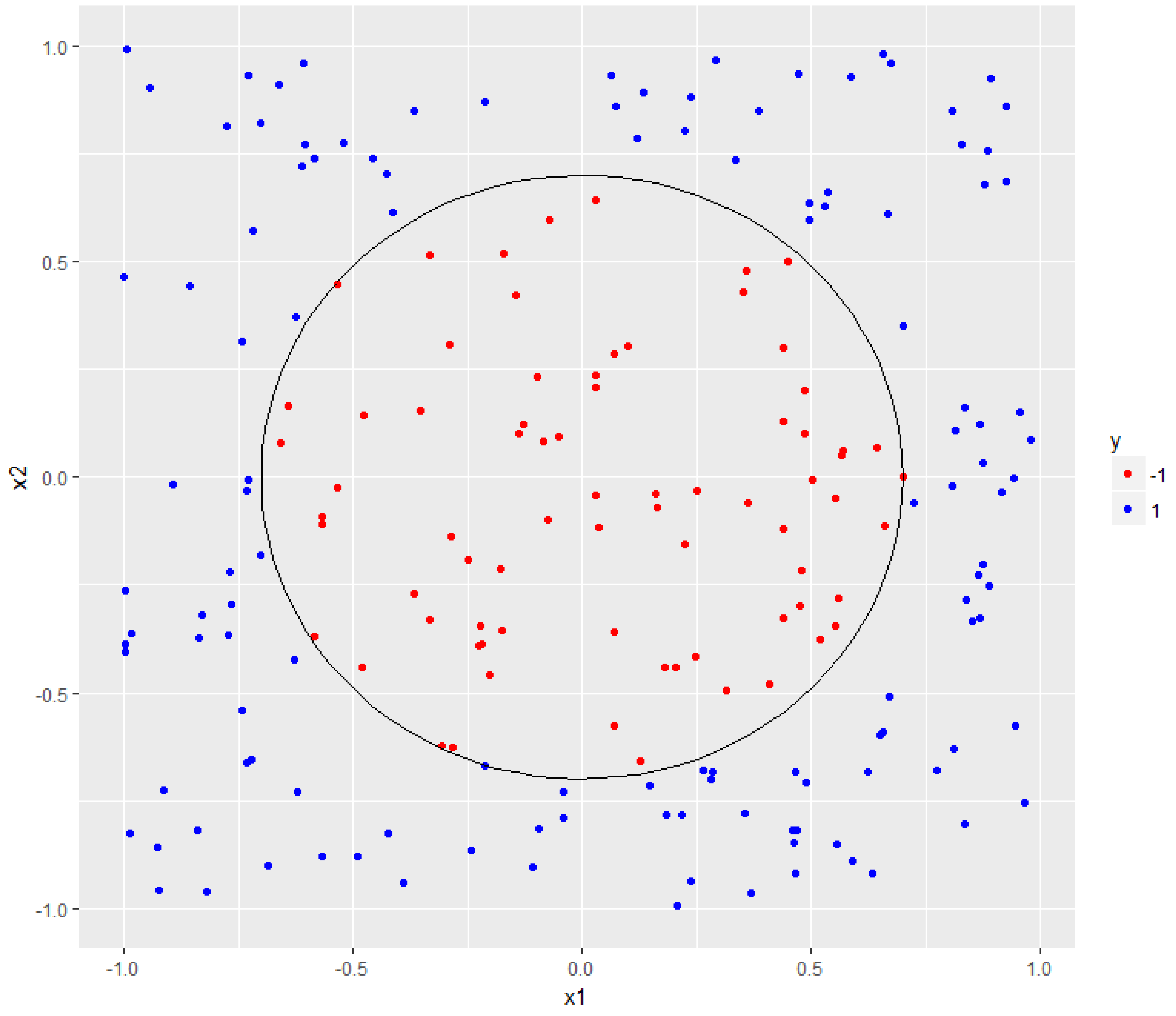
- We'll create a function to generate a circle

```
# 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)

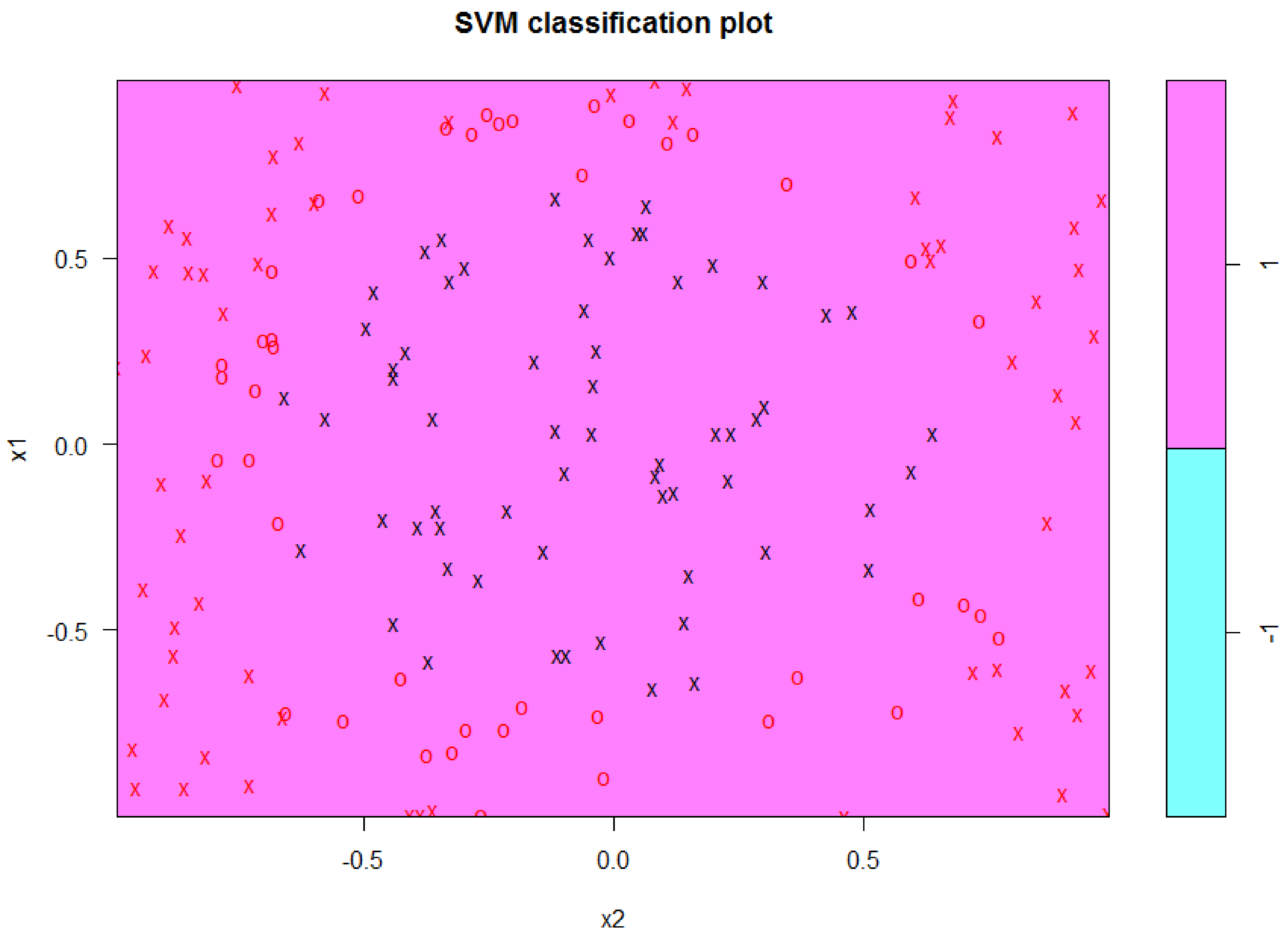
```
# Build default cost linear SVM on training set
svm_model <- svm(y ~ ., data = trainset, type = "C-classification", kernel = "linear", cost = 1)
svm_model
```

```
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
0.5
0.0
-0.5
-0.5
0.0
0.5
x2
1
-1

Linear SVM, cost = 100

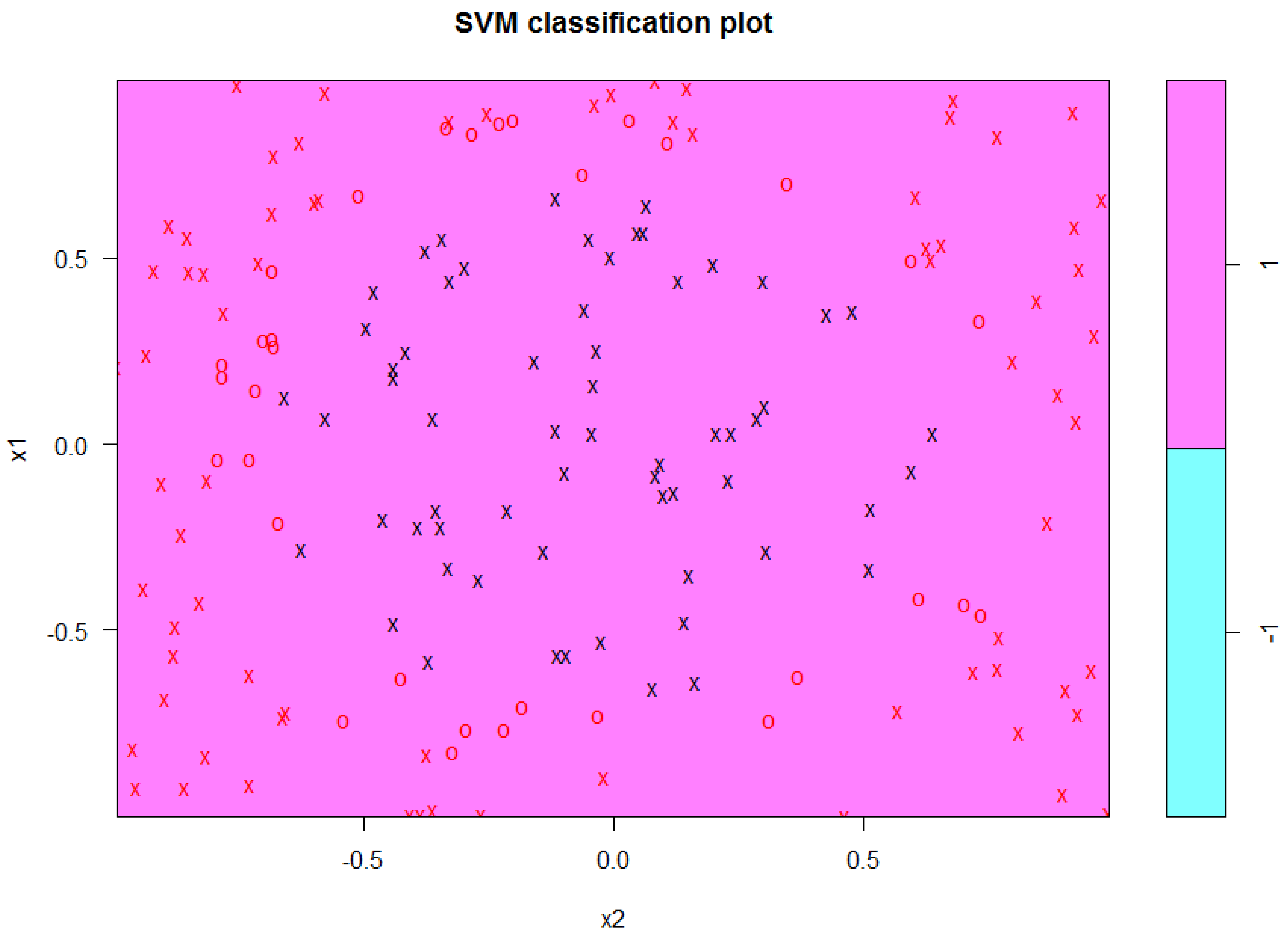
```
svm_model <- svm(y ~ ., data = trainset, type = "C-classification", kernel = "linear", cost = 100)
svm_model
```

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

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

```
0.6129032
```

```
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) {
  sample_size <- floor(0.8 * nrow(df))
  train <- sample(seq_len(nrow(df)), size = sample_size)
  trainset <- df[train, ]
  testset <- df[-train, ]
  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)
```

```
0.544
0.04273184
```

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

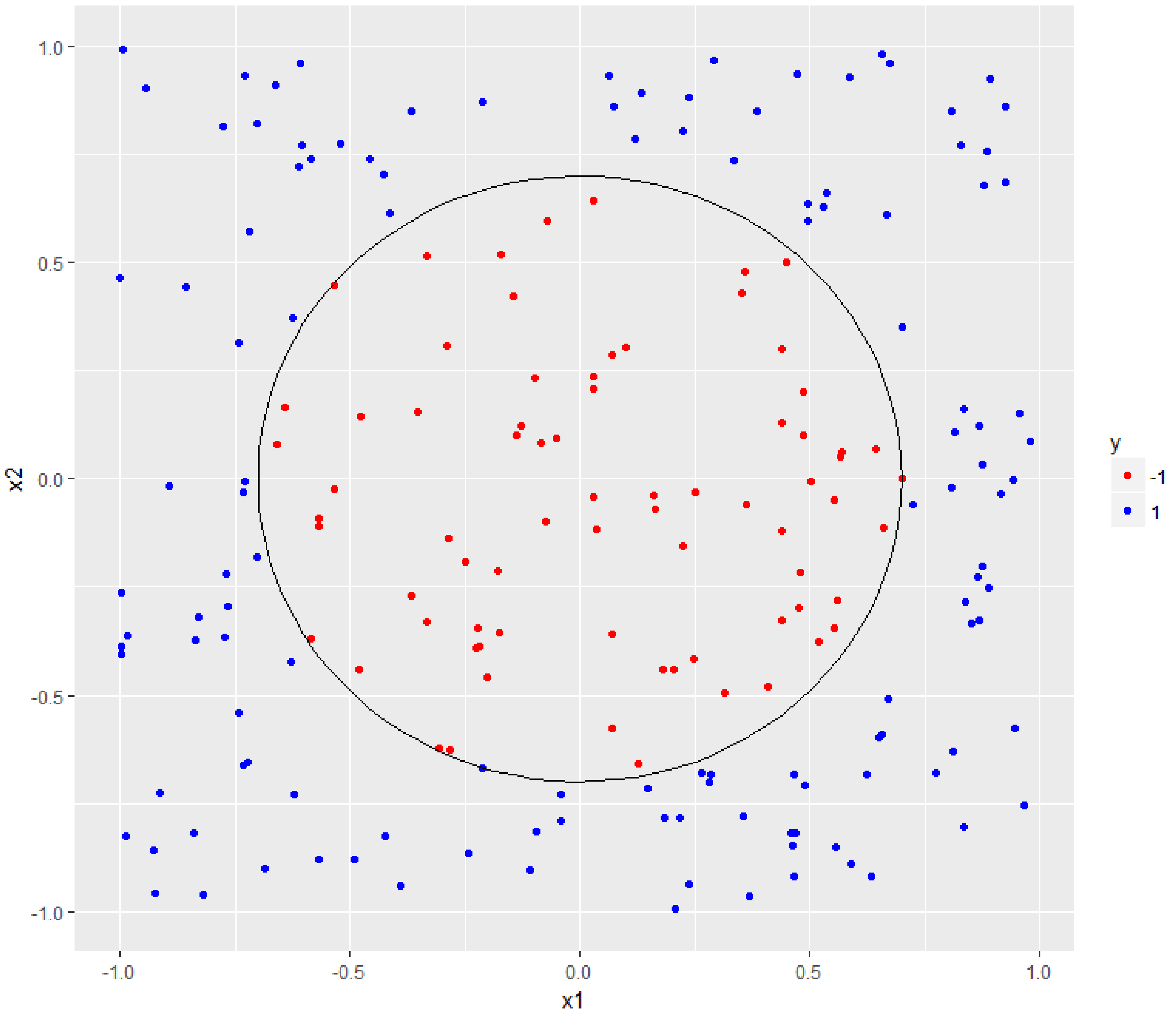
The kernel trick

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

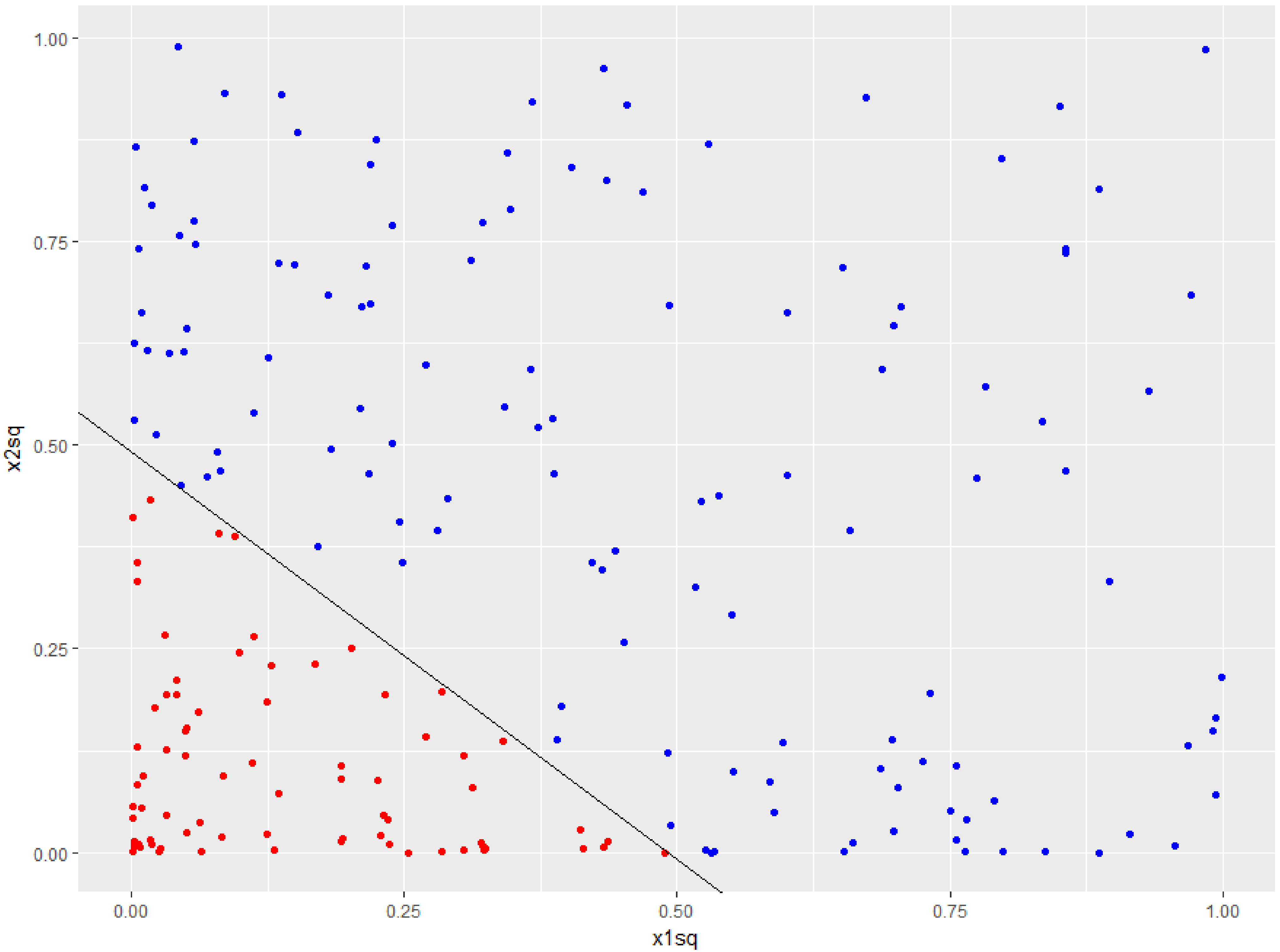
Transforming the problem

- 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!

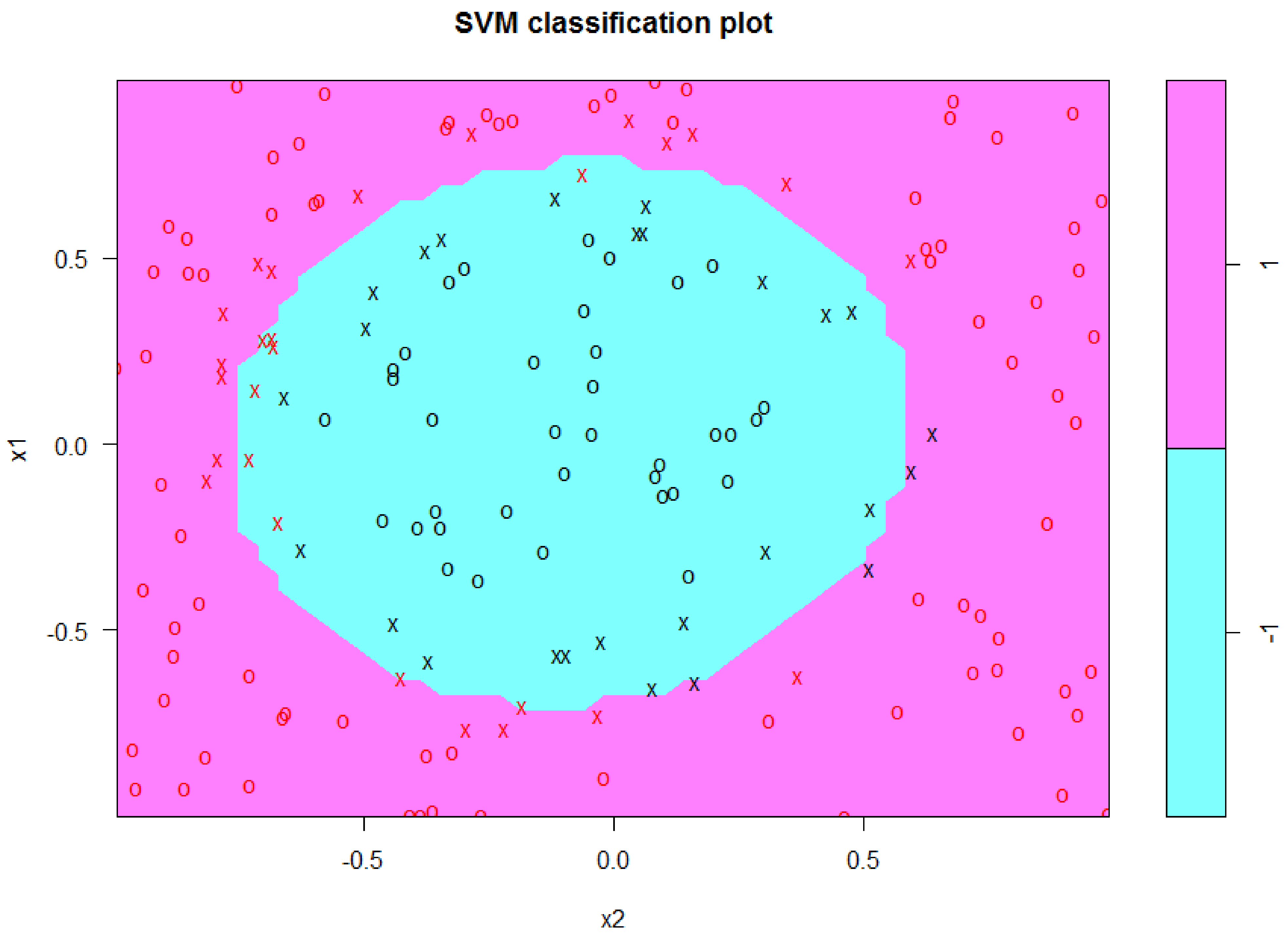
Radially separable dataset - quadratic kernel

- 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
-0.5
0.0
0.5
x2
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()` .

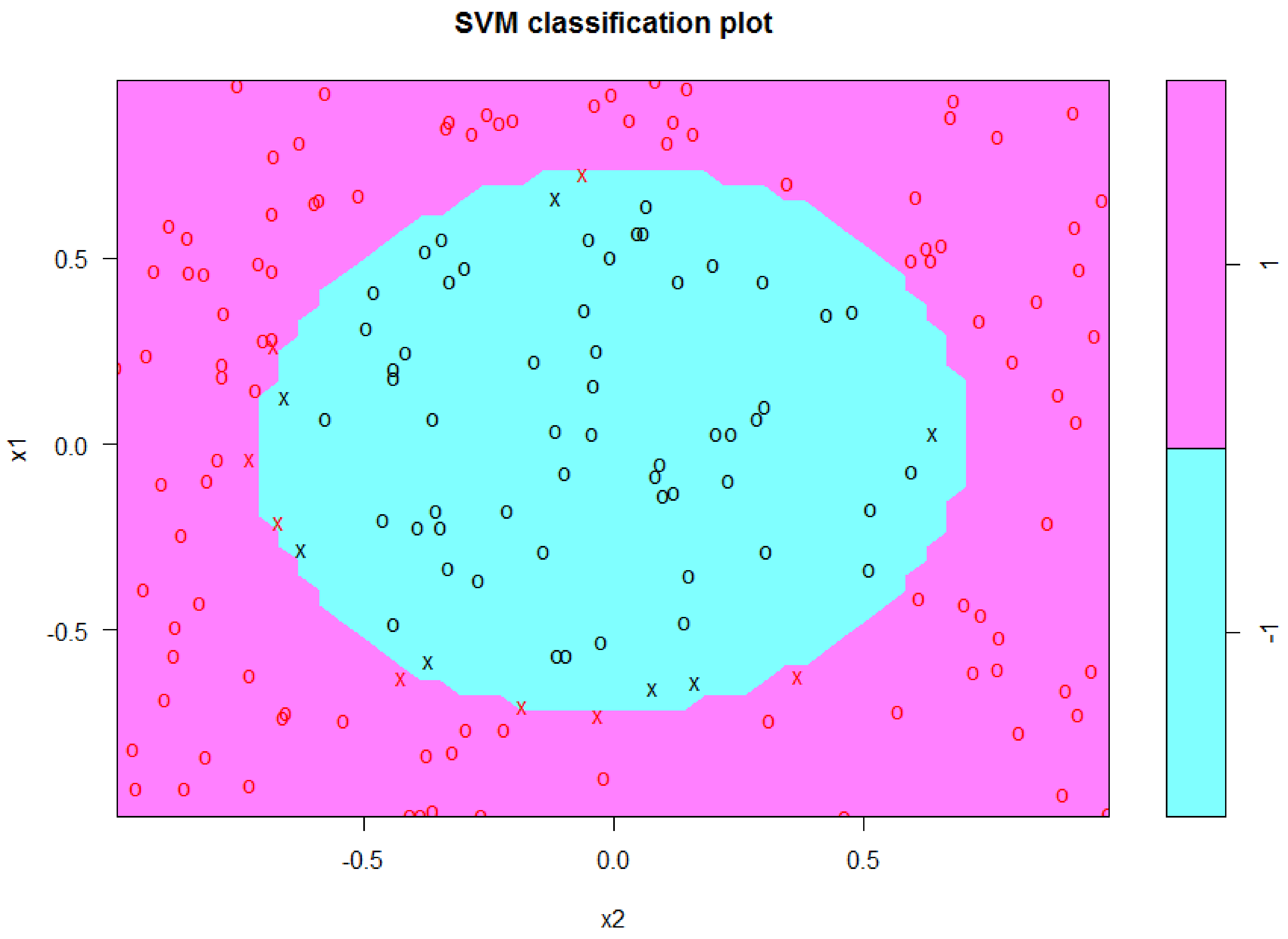
```
svm_model <- svm(y ~ ., data = trainset, type = "C-classification", kernel = "polynomial", degree = 2,
        cost = tune_out$best.parameters$cost,
        gamma = tune_out$best.parameters$gamma,
        coef0 = tune_out$best.parameters$coef0)
```

- 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

RBF Kernels: Generating a complex dataset

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

A bit about RBF Kernels

- Highly flexible kernel.
  - Can fit complex decision boundaries.
- Commonly used in practice.

Generate a complex dataset

- 600 points (x1, x2)
- x1 and x2 distributed differently

```
n <- 600
set.seed(42)

df <- data.frame(x1 = rnorm(n, mean = -0.5, sd = 1),
                 x2 = runif(n, min = -1, max = 1))
```

Generate boundary

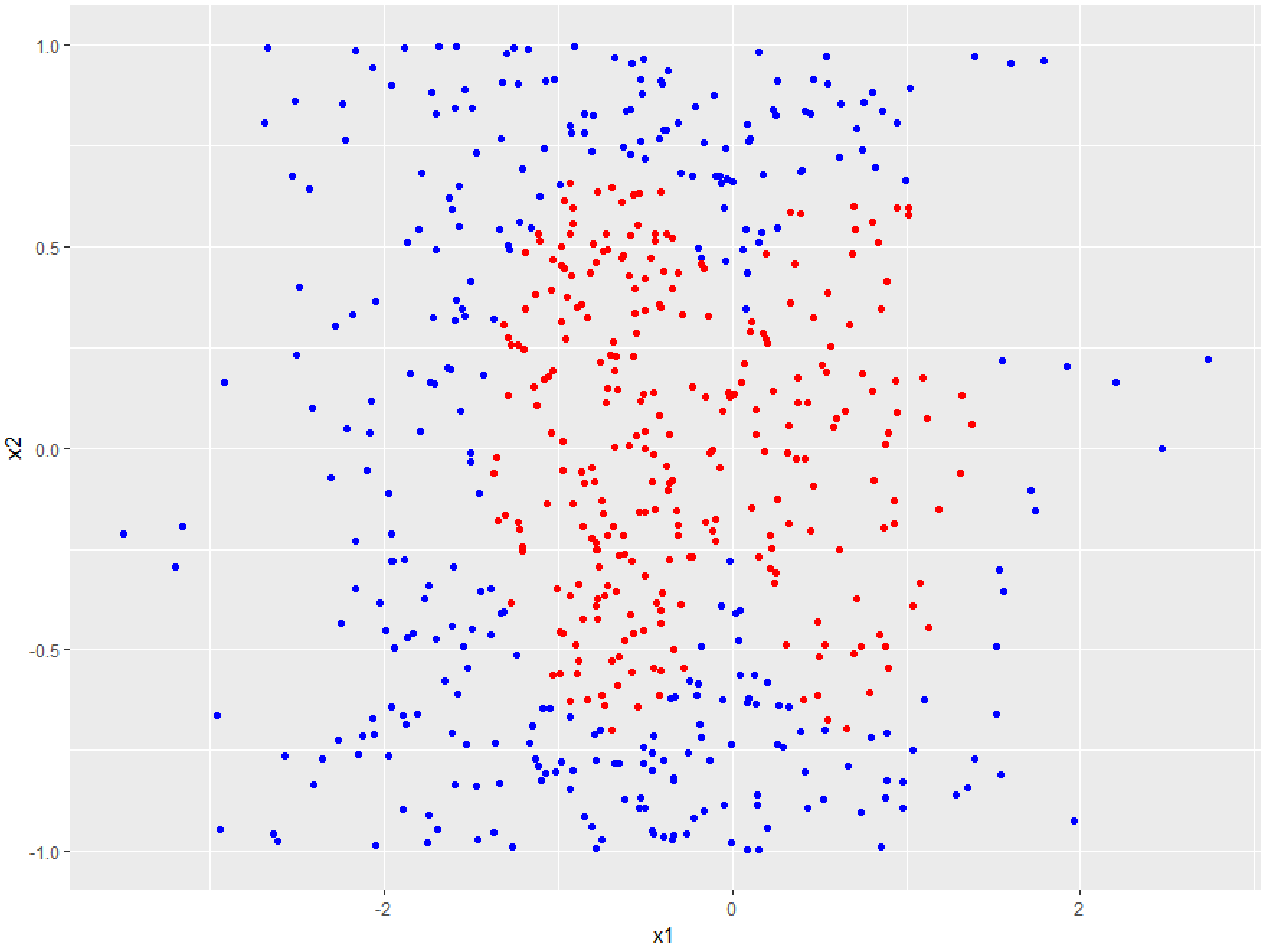
- Boundary consists of two equi-radial circles with a single point in common.

```
# Set radius and centers
radius <- 0.7
radius_squared <- radius ^ 2
center_1 <- c(-0.7, 0)
center_2 <- c(0.7, 0)
# Classify points
df$y <-
    factor(ifelse(
        (df$x1 - center_1[1]) ^ 2 + (df$x2 - center_1[2]) ^ 2 < radius_squared |
        (df$x1 - center_2[1]) ^ 2 + (df$x2 - center_2[2]) ^ 2 < radius_squared,
        -1, 1), levels = c(-1, 1))
```

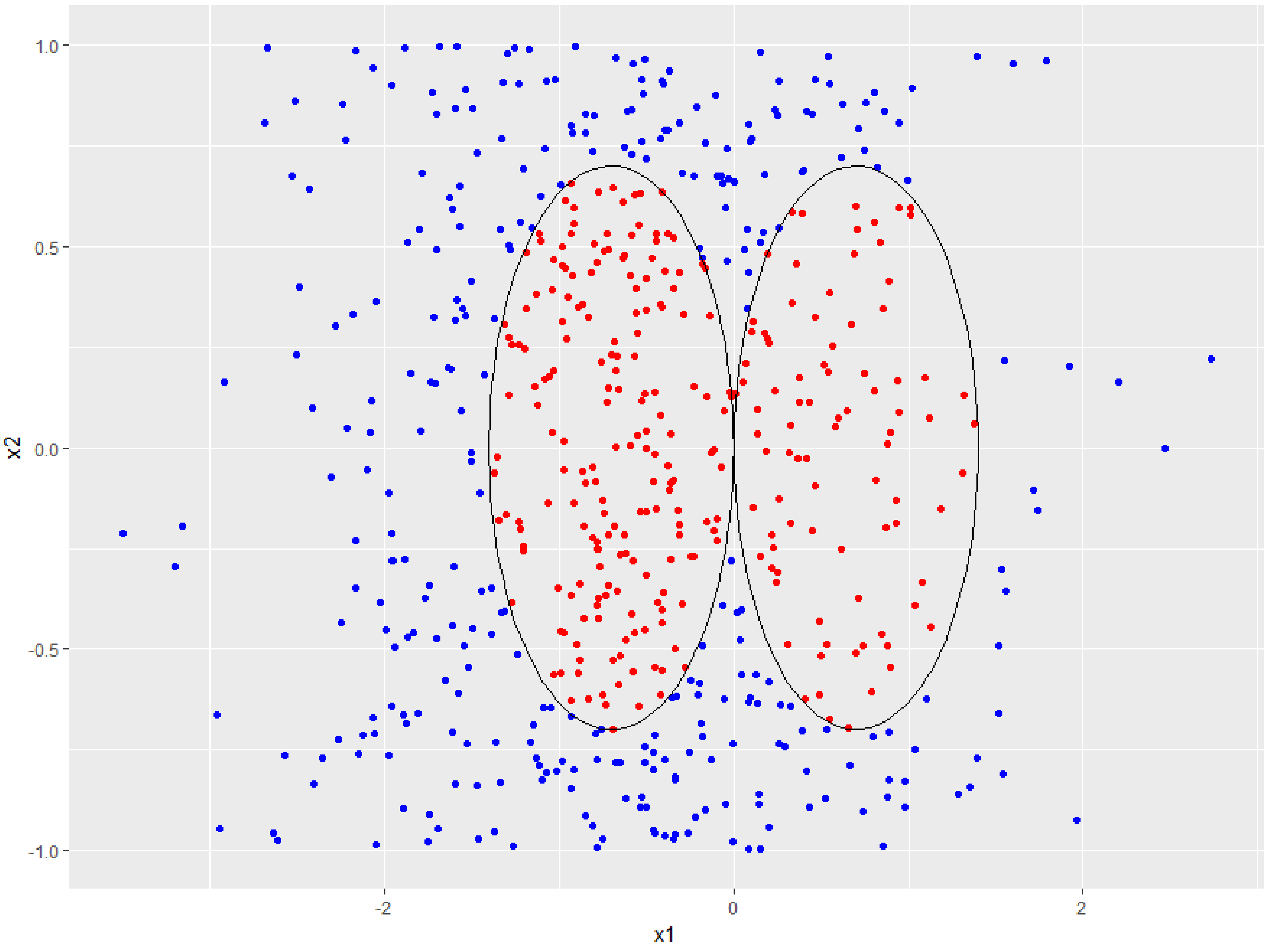
Visualizing the dataset

- Visualize the dataset using ggplot; distinguish classes by color

```
library(ggplot2)

p <- ggplot(data = df, aes(x = x1, y = x2, color = y)) +
      geom_point() +
      guides(color = "none") +
      scale_color_manual(values = c("red", "blue"))

p
```


1.0
0.5
0.0
-0.5
-1.0
x2
-2
0
2
x1

```
# Function to generate points on a circle
circle <- function(x1_center, x2_center, r, npoint = 100) {
   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)
}
# Generate boundary and plot it
boundary_1 <- circle(x1_center = center_1[1], x2_center = center_1[2], r = radius)
p <- p +
     geom_path(data = boundary_1,
               aes(x = x1c, y = x2c),
               inherit.aes = FALSE)
boundary_2 <- circle(x1_center = center_2[1], x2_center = center_2[2], r = radius)
p <- p +
     geom_path(data = boundary_2,
               aes(x = x1c, y = x2c),
               inherit.aes = FALSE)
p
```



1.0
0.5
0.0
-0.5
-1.0
x2
-2
0
2
x1

Time to practice!

SUPPORT VECTOR MACHINES IN R

Motivating the RBF kernel

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

Quadratic kernel (default parameters)

- Partition data into test/train (not shown)
- Use degree 2 polynomial kernel (default params)

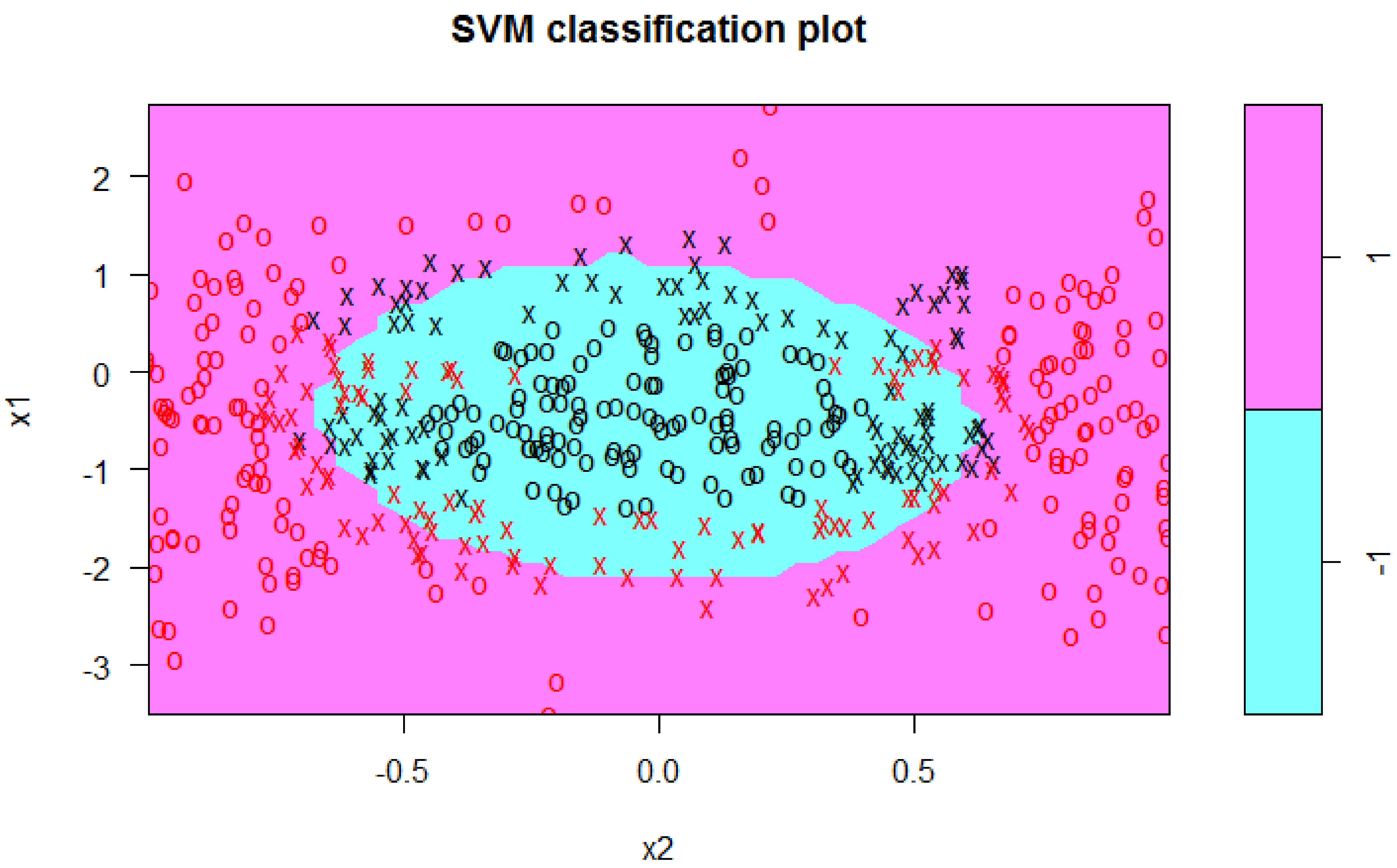
```
svm_model <- svm(y ~ ., data = trainset,
                 type = "C-classification",
                 kernel = "polynomial",
                 degree = 2)
svm_model
```

```
....
Number of Support Vectors:  204
```

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

```
0.8666667
```

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


SVM classification plot
2
1
0
-1
-2
-3
x1
-0.5
0.0
0.5
x2
1
-1

Try higher degree polynomial

- Rule out odd degrees -3,5,9 etc.
- Try degree 4

```
svm_model <- svm(y ~ ., data = trainset,
                 type = "C-classification",
                 kernel = "polynomial",
                 degree = 4)
svm_model
```

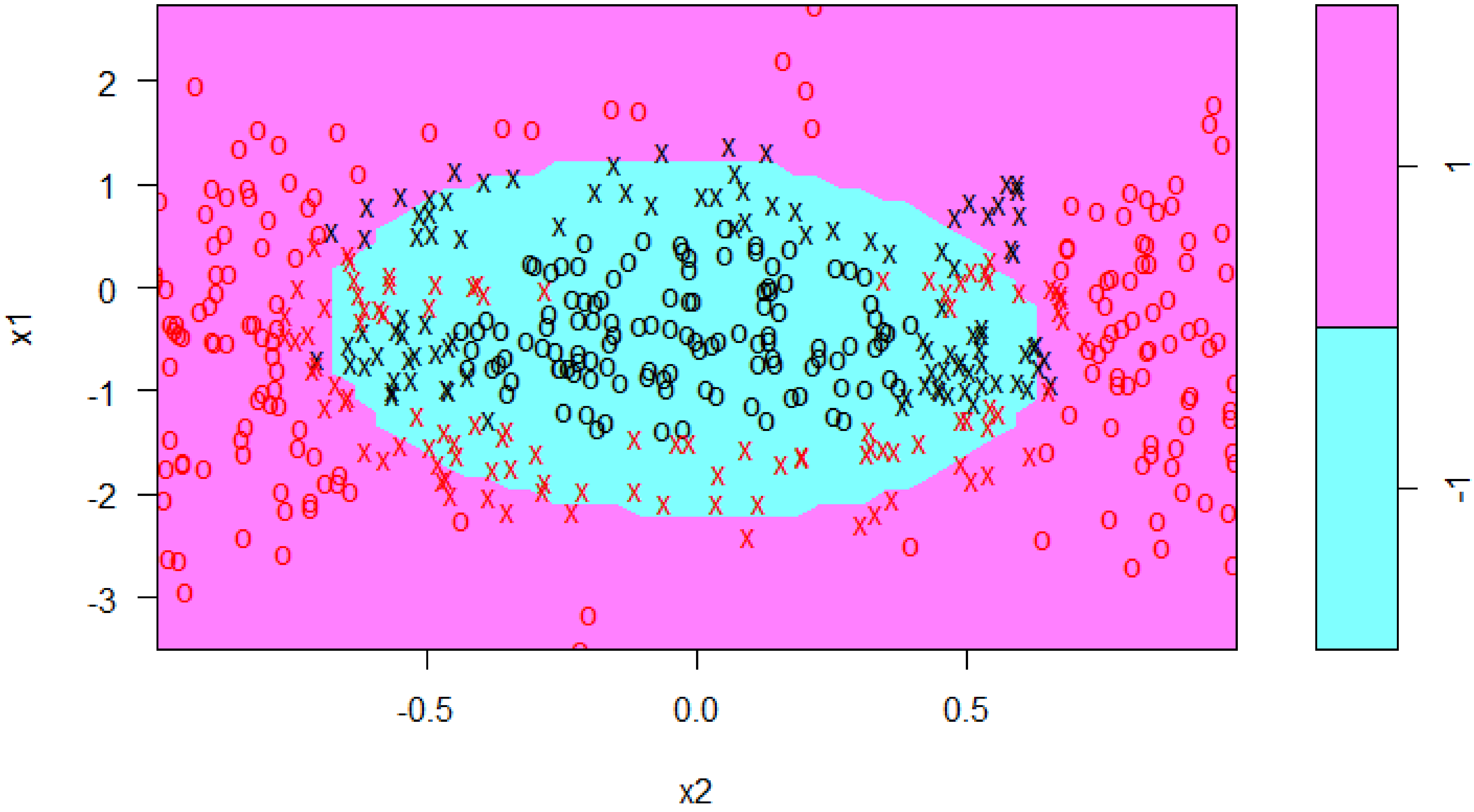
```
...
Number of Support Vectors: 203
```

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

```
0.8583333
```

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

SVM classification plot

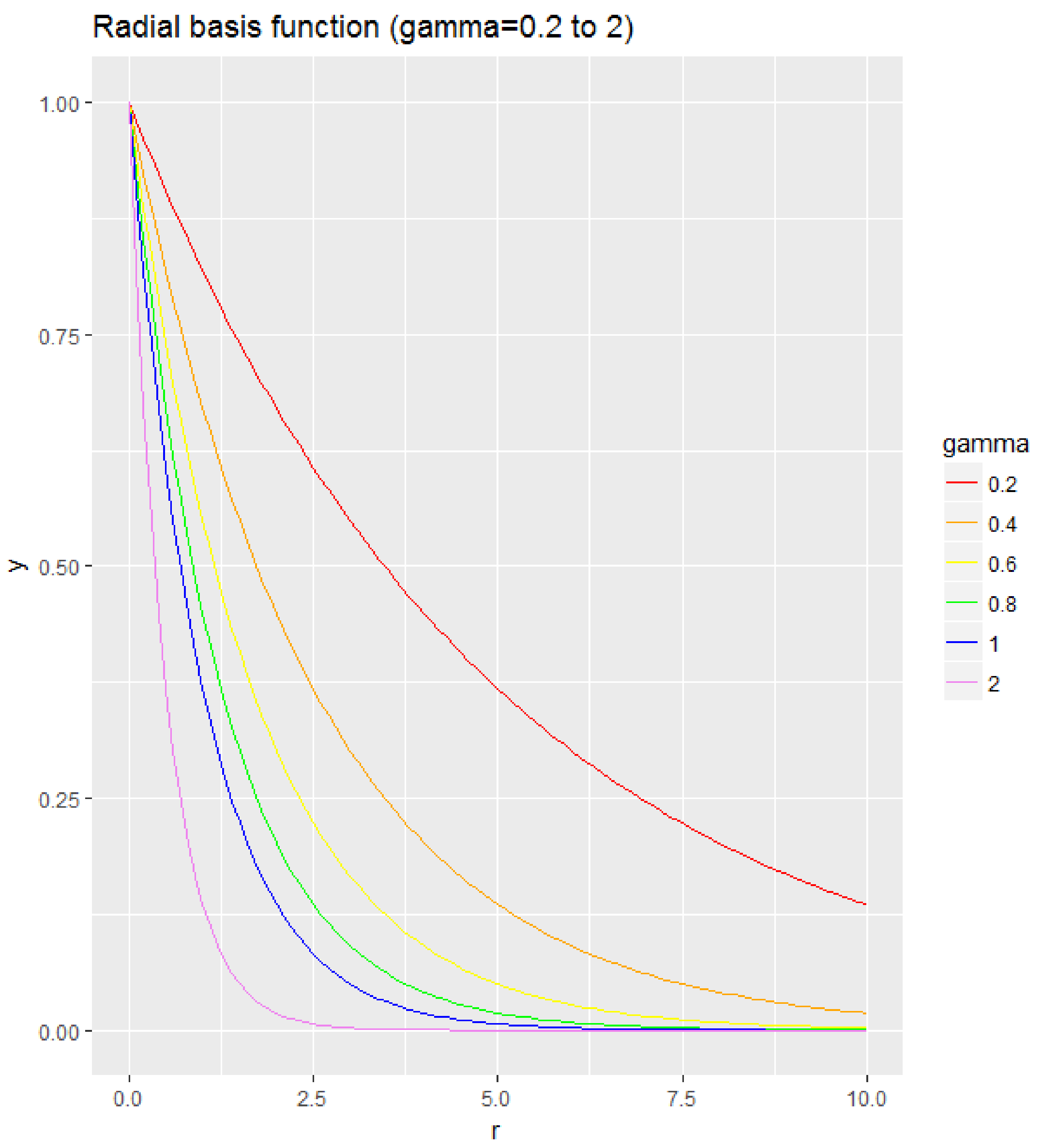


Another approach

- **Heuristic:** points close to each other have the same classification:
  - Akin to K-Nearest Neighbors algorithm.
- For a given point in the dataset, say **X1** = (a, b):
  - The kernel should have a maximum at (a, b)
  - Should decay as one moves away from (a, b)
  - The rate of decay should be the same in all directions
  - The rate of decay should be tunable
- A simple function with this property is `exp(-gamma * r)`, where `r` is the distance between X1 and any other point X

How does the RBF kernel vary with gamma (code)

```
#rbf function
rbf <- function(r, gamma) exp(-gamma * r)
ggplot(data.frame(r = c(-0, 10)), aes(r)) +
  stat_function(fun = rbf, args = list(gamma = 0.2), aes(color = "0.2")) +
  stat_function(fun = rbf, args = list(gamma = 0.4), aes(color = "0.4")) +
  stat_function(fun = rbf, args = list(gamma = 0.6), aes(color = "0.6")) +
  stat_function(fun = rbf, args = list(gamma = 0.8), aes(color = "0.8")) +
  stat_function(fun = rbf, args = list(gamma = 1), aes(color = "1")) +
  stat_function(fun = rbf, args = list(gamma = 2), aes(color = "2")) +
  scale_color_manual("gamma",
                     values = c("red","orange","yellow",
                                "green","blue","violet")) +
  ggtitle("Radial basis function (gamma = 0.2 to 2)")
```



Radial basis function (gamma=0.2 to 2)
1.00
0.75
0.50
0.25
0.00
y
0.0
2.5
5.0
7.5
10.0
r
gamma
0.2
0.4
0.6
0.8
1
2

Time to practice!

SUPPORT VECTOR MACHINES IN R

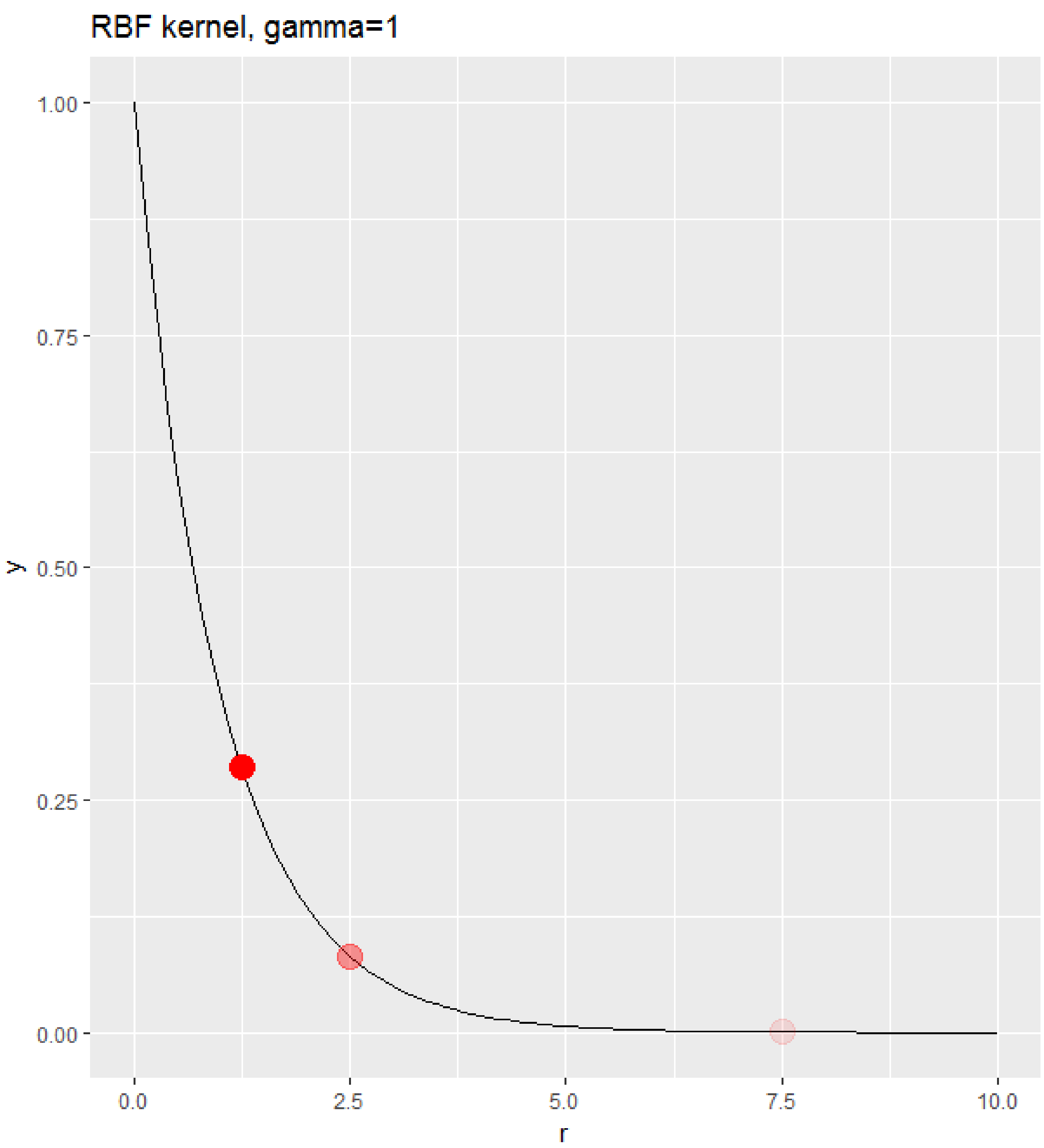
The RBF Kernel

SUPPORT VECTOR MACHINES IN R

Kailash Awati
Instructor

RBF Kernel in a nutshell

- Decreasing function of distance between two points in dataset.
- Simulates k-NN algorithm.


RBF kernel, gamma=1
1.00
0.75
0.50
0.25
0.00
y
0.0
2.5
5.0
7.5
10.0
r

Building an SVM using the RBF kernel

- Build RBF kernel SVM for complex dataset

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

- Calculate training/test accuracy and plot against training dataset.

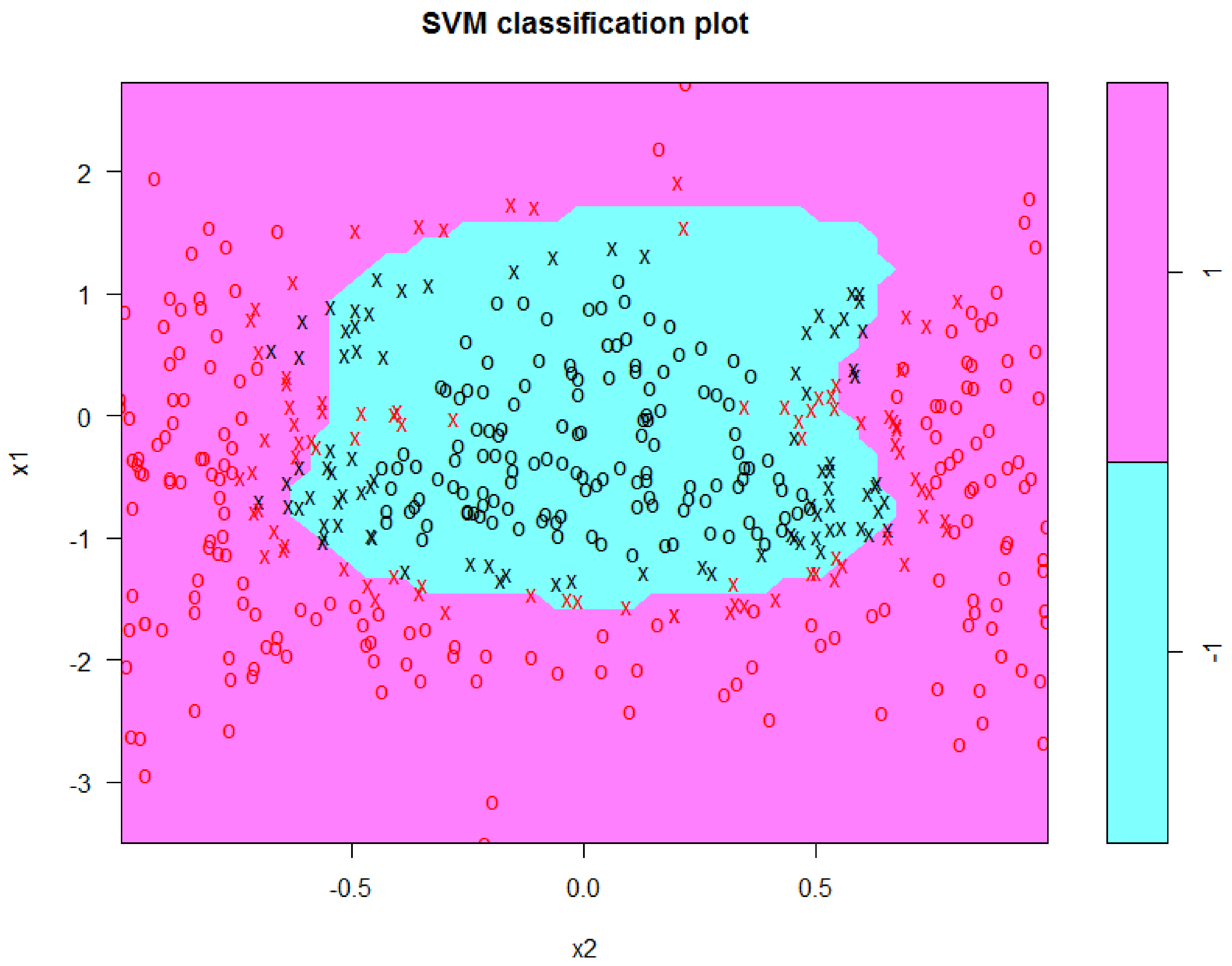
```
pred_train <- predict(svm_model, trainset)
mean(pred_train == trainset$y)
```

```
0.93125
```

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

```
0.9416667
```

```
#plot decision boundary
plot(svm_model, trainset)
```


SVM classification plot
2
1
0
-1
-2
-3
x1
-0.5
0.0
0.5
x2
1
-1

Refining the decision boundary

- Tune `gamma` and `cost` using `tune.svm()`

```
# Tune parameters
tune_out <- tune.svm(x = trainset[,-3],
                     y = trainset[,3],
                     gamma = 5 * 10 ^ (-2:2),
                     cost = c(0.01, 0.1, 1, 10, 100
                     type = "C-classification",
                     kernel = "radial")
```

- Print best parameters

```
# Print best values of cost and gamma
tune_out$best.parameters$cost
```

```
1
```

```
tune_out$best.parameters$gamma
```

```
5
```

The tuned model

- Build tuned model using `best.parameters`

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

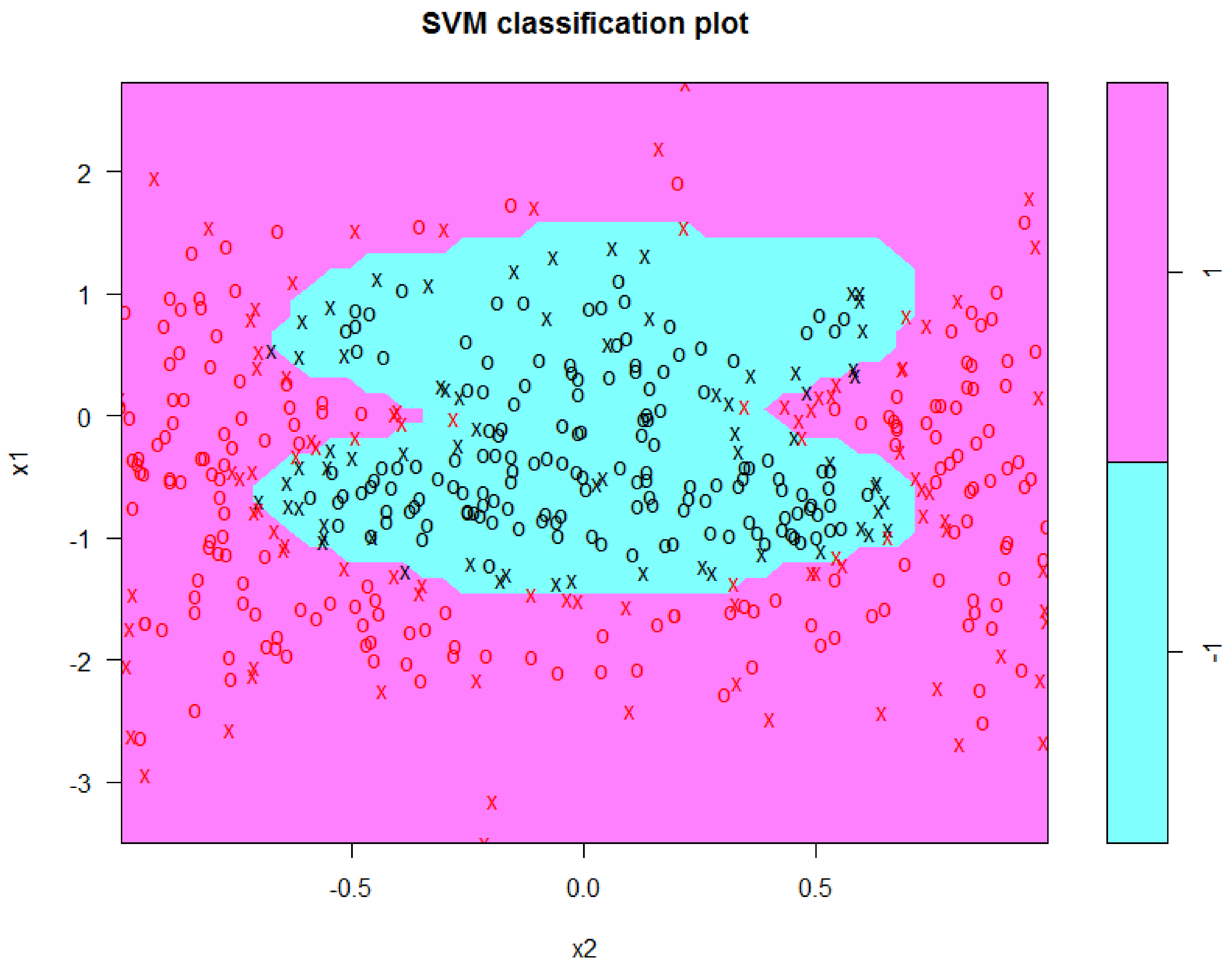
- Calculate test accuracy

```
mean(pred_test == testset$y)
```

```
0.95
```

- Plot decision boundary

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


SVM classification plot
x1
x2
2
1
0
-1
-2
-3
-0.5
0.0
0.5
1
-1

Time to practice!

SUPPORT VECTOR MACHINES IN R