



SUPPORT VECTOR MACHINES IN R

Generating a radially separable dataset



Generating a 2d uniformly distributed set of points

- Generate a dataset with 200 points
 - 2 predictors x_1 and x_2 , 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))
```

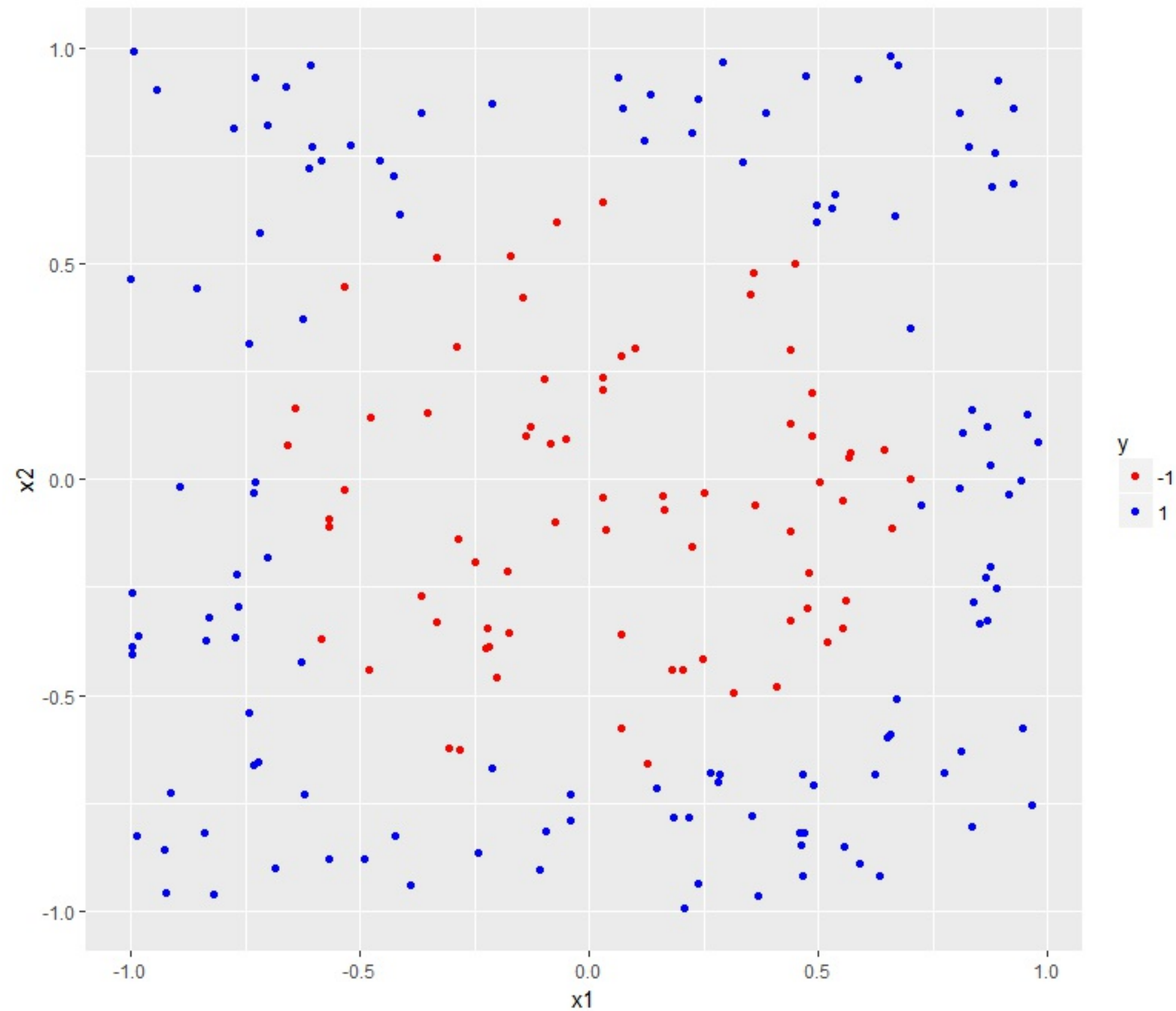
Plot the dataset

- Visualize using ggplot.

```
#load 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
```



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)

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

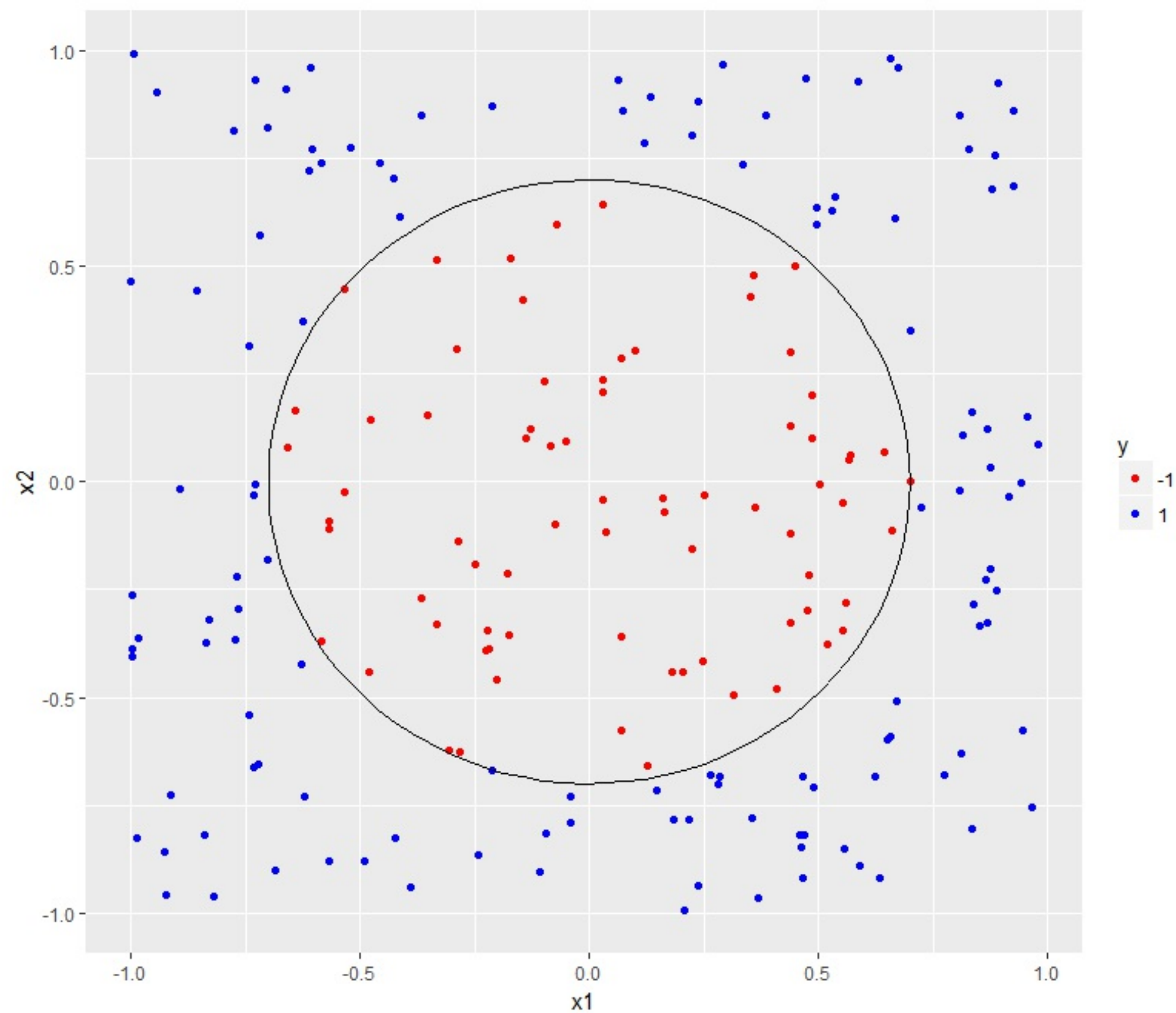
Adding a circular boundary - Part 2

- 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
```





SUPPORT VECTOR MACHINES IN R

Time to practice!



SUPPORT VECTOR MACHINES IN R

Linear SVMs on radially separable data

Linear SVM, cost = 1

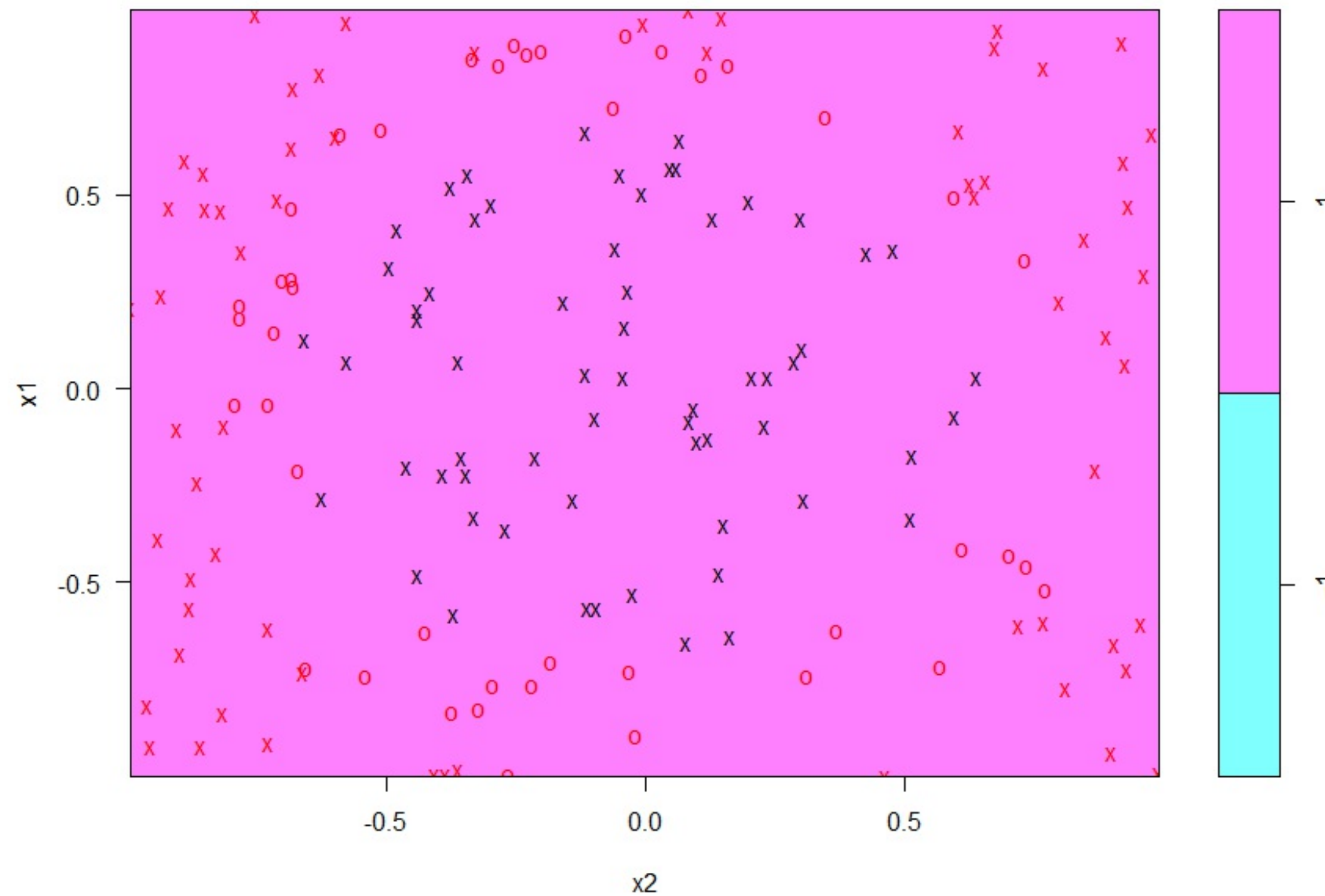
- Partition radially separable dataset into training/test (seed = 10)
- Build default cost linear SVM on training set

```
svm_model<-  
  svm(y ~ ., data=trainset, type="C-classification", kernel="linear")  
svm_model  
.....  
Number of Support Vectors: 126
```

- Calculate accuracy on test set.

```
#accuracy  
pred_test <- predict(svm_model, testset)  
mean(pred_test == testset$y)  
[1] 0.6129032  
  
#plot  
plot(svm_model, trainset)
```

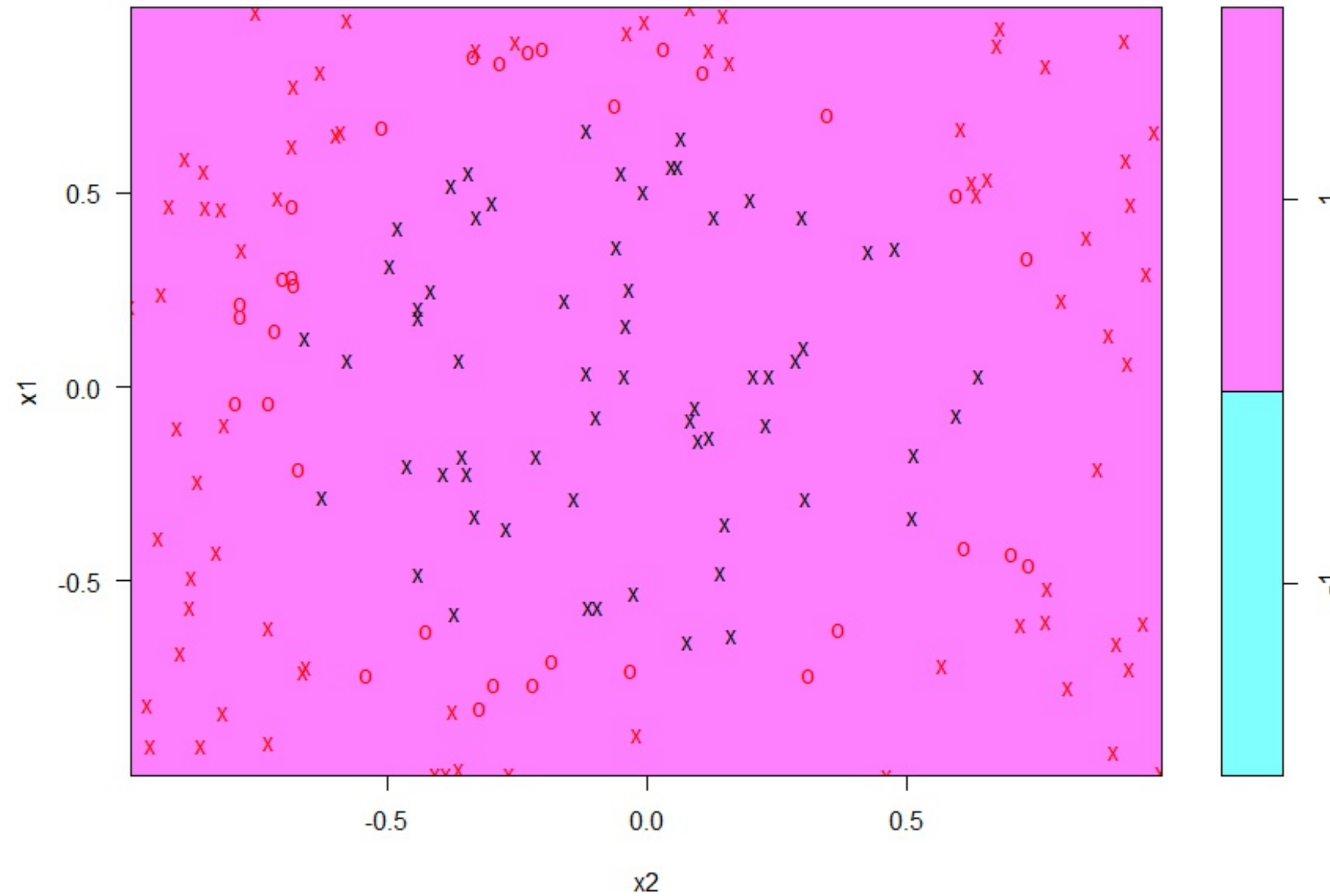
SVM classification plot



Linear SVM, cost = 100

```
svm_model<-  
  svm(y ~ ., data=trainset, type="C-classification", kernel="linear")  
svm_model  
.....  
Number of Support Vectors: 136  
#accuracy  
pred_test <- predict(svm_model,testset)  
mean(pred_test==testset$y)  
[1] 0.6129032  
plot(svm_model,trainset)
```

SVM classification plot





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)
[1] 0.642843
sd(accuracy)
[1] 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.



SUPPORT VECTOR MACHINES IN R

Time to practice!



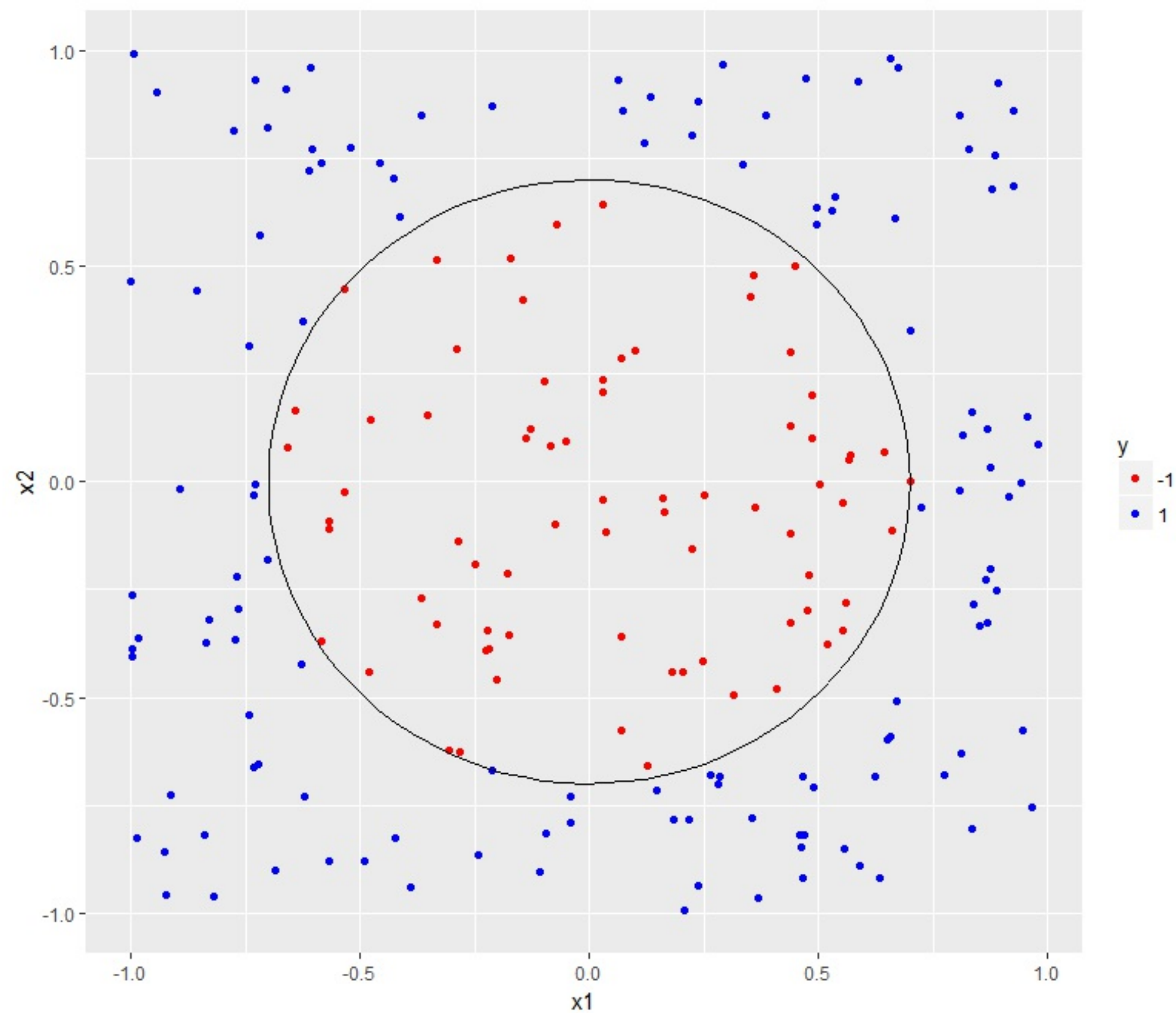
SUPPORT VECTOR MACHINES IN R

The Kernel Trick



The basic idea

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





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!!)

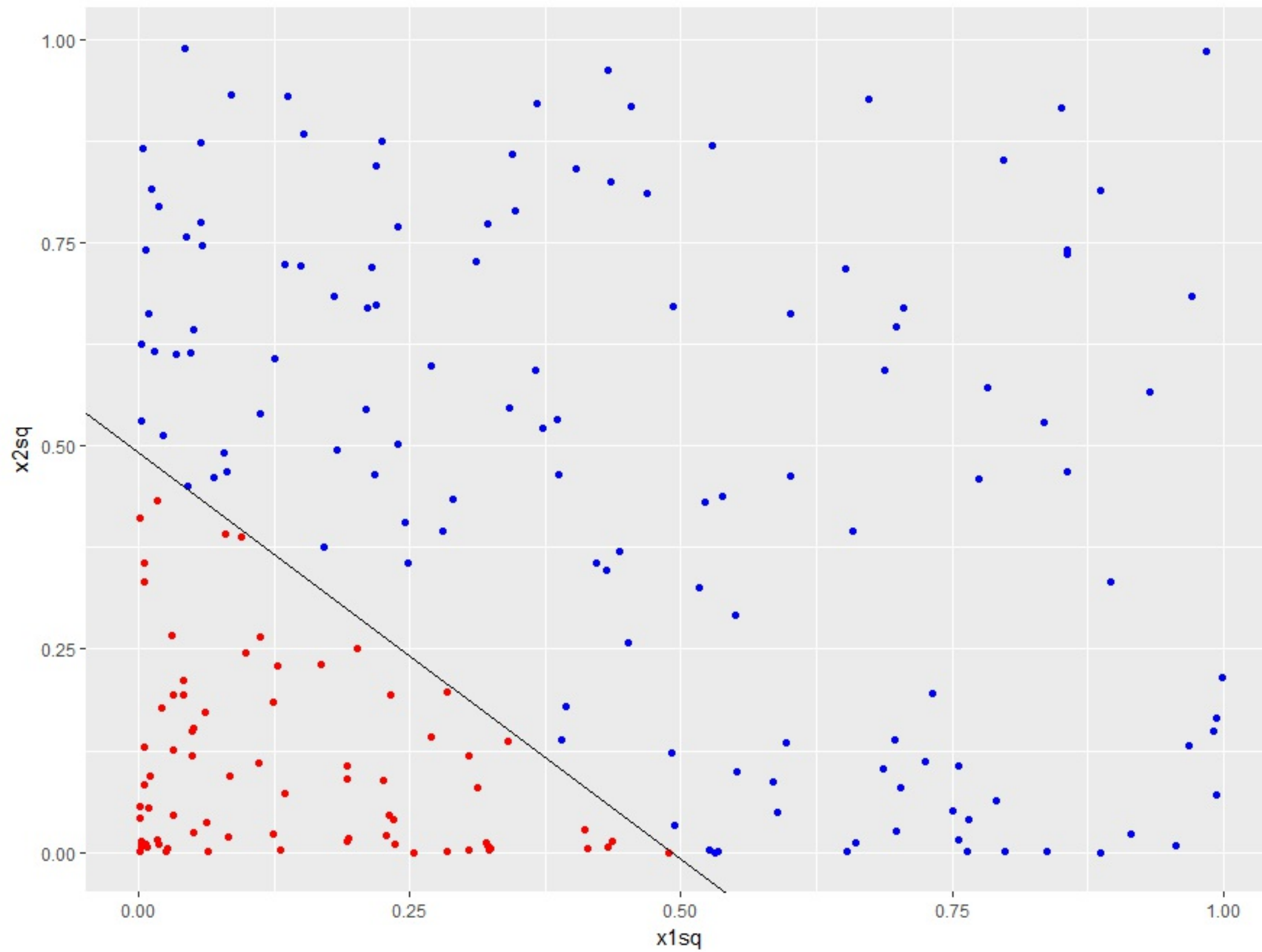


Plot in X1-X2 space - code

- Use ggplot() to plot the dataset in X1-X2 space
- Equation of boundary $X2 = -X1 + 0.49$:
 - slope=-1
 - y-intercept=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



The Polynomial Kernel - Part 1

- Polynomial kernel: $(\gamma \cdot (u \cdot v) + \text{coef0})^{\text{degree}}$
 - degree = degree of polynomial
 - gamma and coef0- tuning parameters
 - u, v - vectors (datapoints) belonging to the dataset
- We can guess we need a 2nd degree polynomial (transformation)



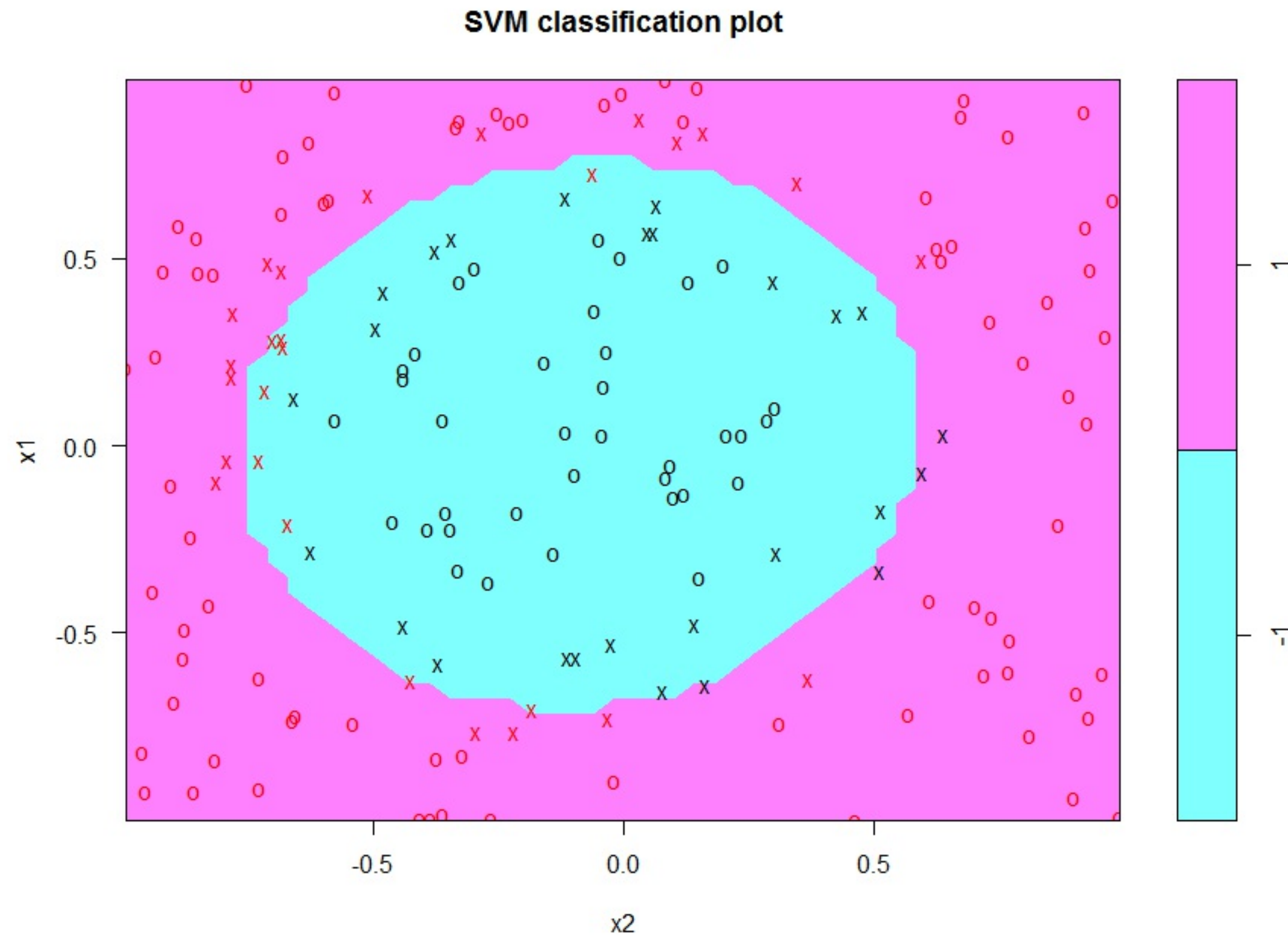
Kernel functions

- 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:
 - degree = 2
 - 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)  
[1] 0.9354839  
  
#visualize model  
plot(svm_model, trainset)
```





SUPPORT VECTOR MACHINES IN R

Time to practice!



SUPPORT VECTOR MACHINES IN R

Tuning SVMs



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: $\text{cost} = 10^{(-1:3)}$, $\text{gamma} = c(0.1, 1, 10)$, $\text{coef0} = c(0.1, 1, 10)$
 - 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!

Introducing tune.svm()

- 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  
[1] 0.1  
tune_out$best.parameters$gamma  
[1] 10  
tune_out$best.parameters$coef0  
[1] 1
```

Build and examine optimal model

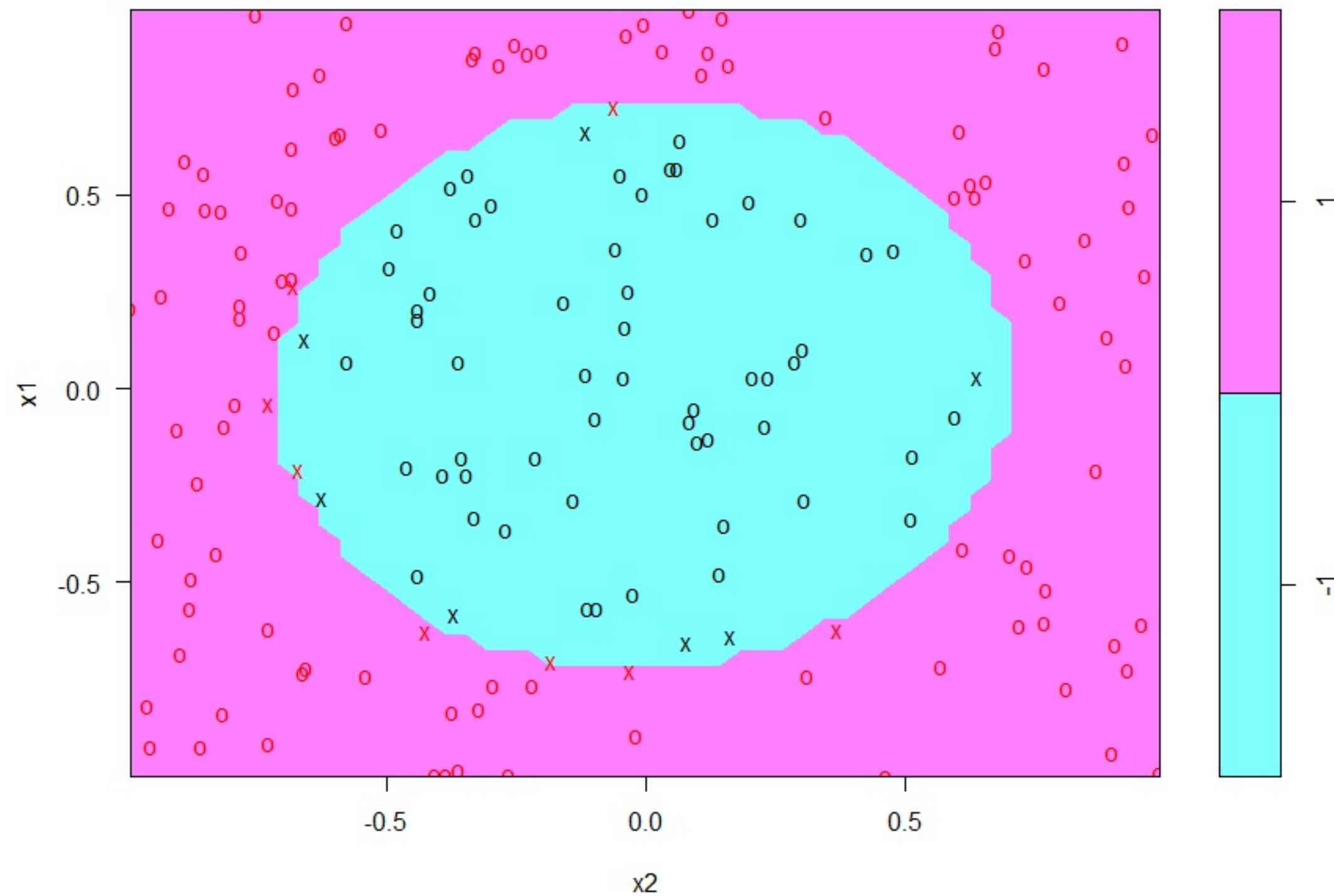
- Build SVM model using best values of parameters from `tune.svm()`.

```
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$y)  
[1] 1  
pred_test <- predict(svm_model, testset)  
mean(pred_test==testset$y)  
[1] 0.9677419  
  
#plot using svm plot  
plot(svm_model, trainset)
```

SVM classification plot





SUPPORT VECTOR MACHINES IN R

Time to practice!