STA6933: Advanced Topics in Statistical Learning

Chapter 6: Support Vector Machines

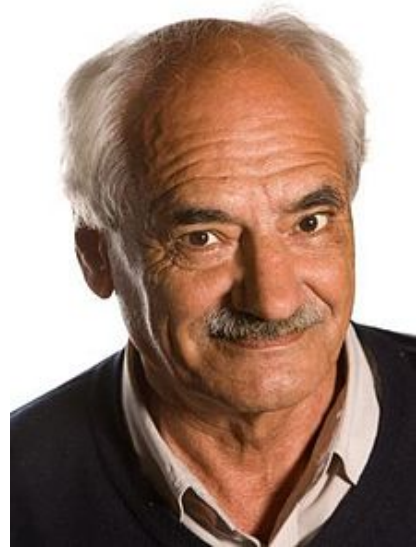
Min Wang

Department of Management Science and Statistics
The University of Texas at San Antonio

min.wang3@utsa.edu

Support Vector Machines

- Everywhere there is a mention of SVM inventors as V. N. Vapnik, A. Y. Chervonenkis in 1963

- The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes. The current standard incarnation (soft margin) was proposed by Corinna Cortes and Vapnik in 1993 ... — Wikipedia

SVMs: a new generation of learning algorithms

- Pre 1980:
  - Almost all learning methods learned linear decision surfaces
  - Linear learning methods have nice theoretical properties
- 1980's
  - Decision trees and NNs allowed efficient learning of nonlinear decision surfaces
  - Little theoretical basis and all suffer from local minima
- 1990's
  - Efficient learning algorithms for non-linear functions based on computational learning theory developed
  - Nice theoretical properties

Key ideas

- Two independent developments within last decade
  - New efficient separability of non-linear regions that use "kernel functions": generalization of "similarity" to new kinds of similarity measures based on dot products
  - Use of quadratic optimization problem to avoid "local minimum" issues with neural nets
  - The resulting learning algorithm is an optimization algorithm rather than a greedy search

Support vector machines

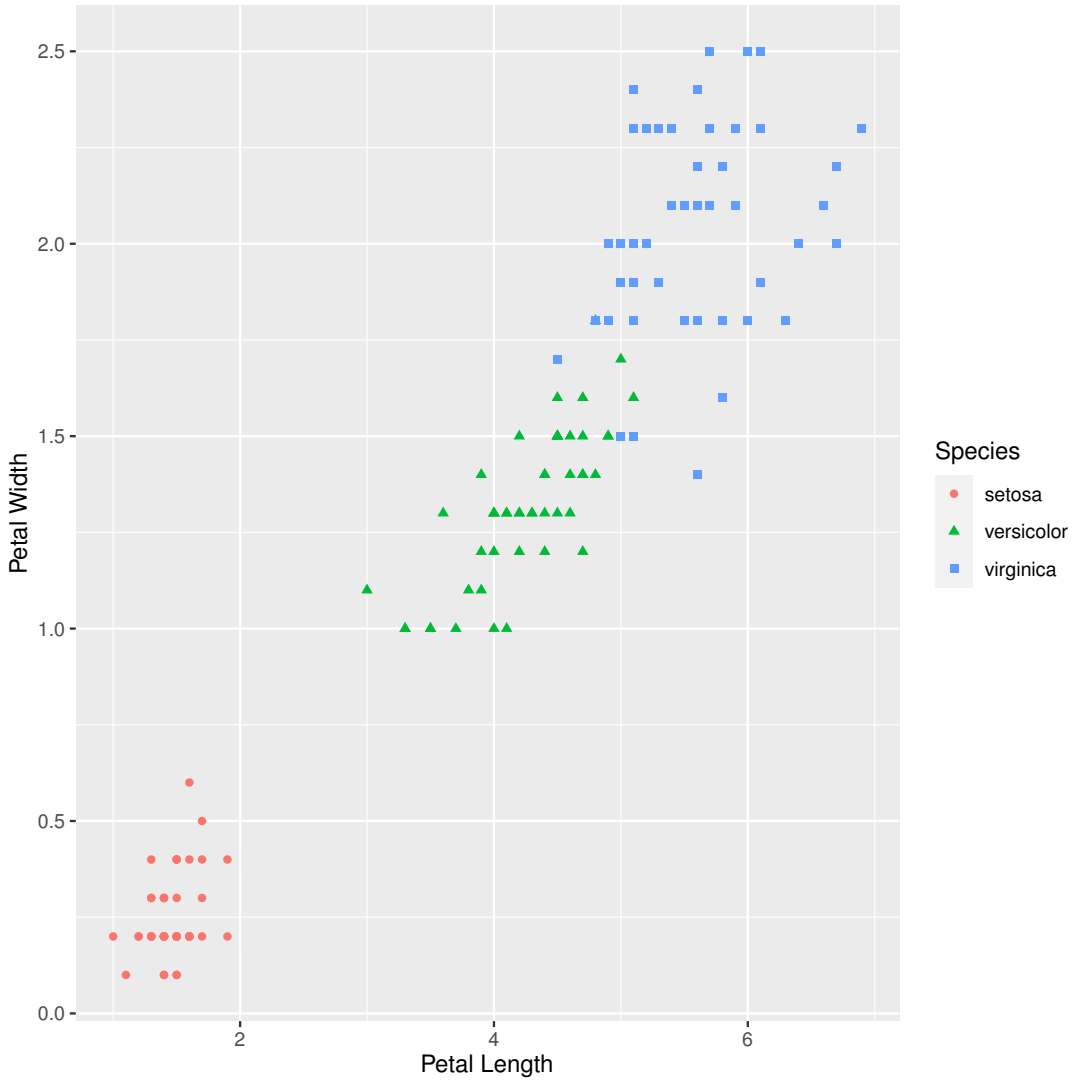
- Support vector machine was developed in the computer science community in the 1990s, and it has been considered one of the best "out of the box" classifiers. Specifically, we will learn
  - maximal margin classifier
  - support vector classifier
  - support vector machine

Support vector machines

- Support vector machine was developed in the computer science community in the 1990s, and it has been considered one of the best "out of the box" classifiers. Specifically, we will learn
  - maximal margin classifier
  - support vector classifier
  - support vector machine
- People often loosely refer to these approaches as "support vector machines." To avoid confusion, we will carefully distinguish between these three notions.

Outline

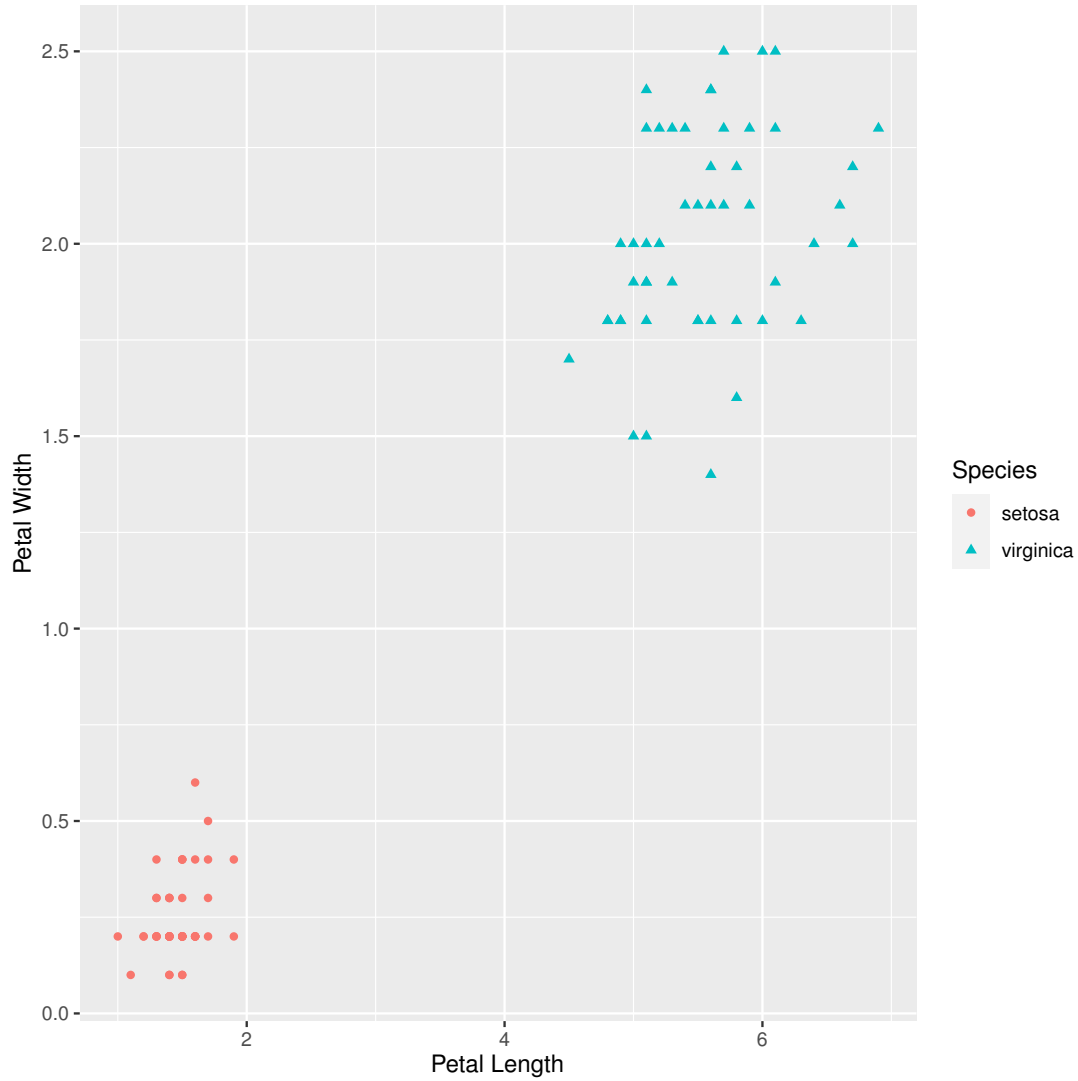
Iris data


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

R code for the previous plot

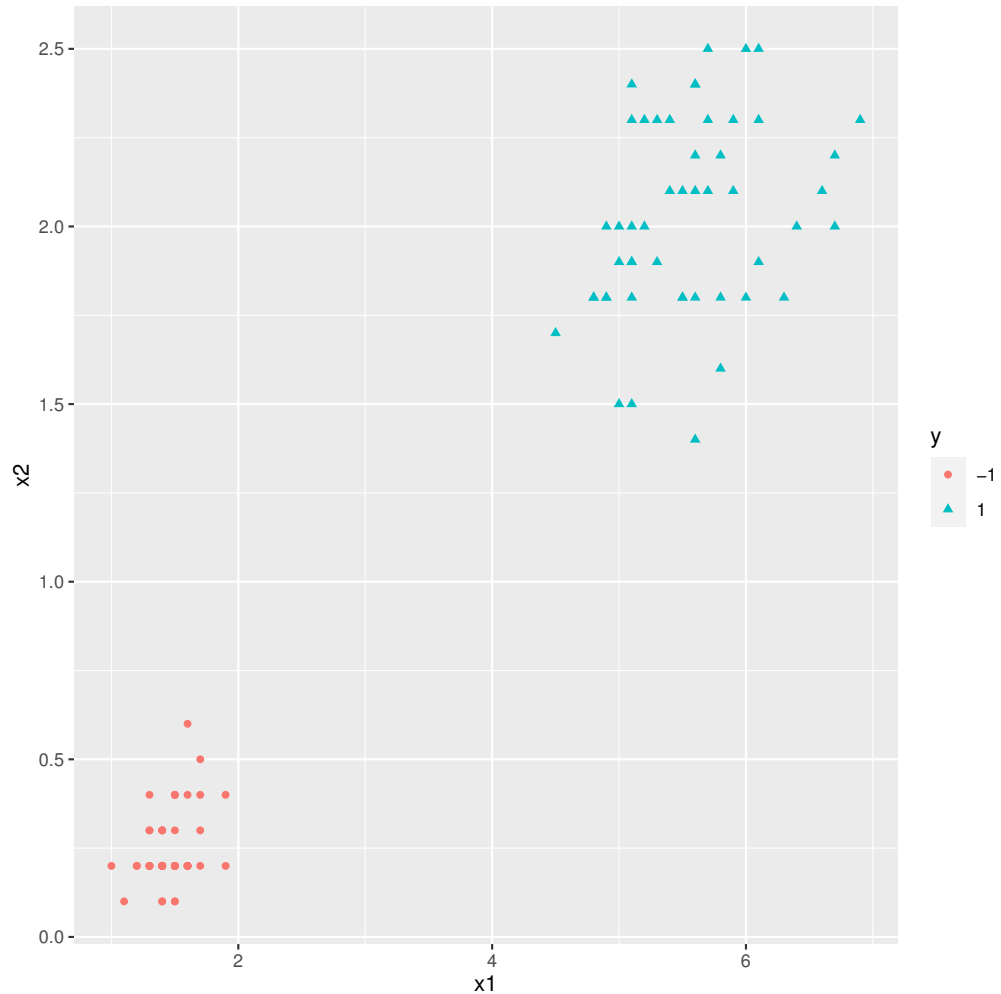
```
library(e1071)
library(ggplot2)
library(gbm)
library(randomForest)
library(scatterplot3d)

#Iris data
par(mfrow = c(1,2))
ggplot(data=iris, aes(x = Petal.Length, y = Petal.Width)) +
  geom_point(aes(color=Species, shape=Species)) +
  xlab("Petal Length") +  ylab("Petal Width")
```

Consider only Species “setosa” and “virginica”



Generalize data

- Assign “-1” to “setosa” and “+1” to “virginica”
- x_1 is “Petal.Length” and x_2 is “Petal.Width”



R code for the previous plot

```
## Consider only Species "setosa" and "virginica"
ggplot(data=iris[iris$Species != "versicolor",], aes(x = Petal.Length, y = Petal.Width)) +
  geom_point(aes(color=Species, shape=Species)) +
  xlab("Petal Length") +  ylab("Petal Width")

## Generalize data
n.iris = iris[iris$Species != "versicolor",c("Petal.Length","Petal.Width","Species")]
n.iris$Species = as.factor(ifelse(n.iris$Species == "setosa",-1,1))
colnames(n.iris) = c("x1","x2","y")
ggplot(data=n.iris, aes(x = x1, y = x2)) + geom_point(aes(color=y, shape=y)) +
  xlab("x1") +  ylab("x2")
```

What is a Hyperplane?

- The mathematical definition of a *hyperplane* is quite simple. In general, a p-dimensional hyperplane is defined by

$$f(x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p = 0,$$

or equivalently $\beta_0 + \beta^T x = 0$, where $\beta = (\beta_1, \ldots, \beta_p)^T$ and $x = (x_1, \ldots, x_p)^T$.

What is a Hyperplane?

- The mathematical definition of a *hyperplane* is quite simple. In general, a p-dimensional hyperplane is defined by

$$f(x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p = 0,$$

  or equivalently $\beta_0 + \beta^T x = 0$, where $\beta = (\beta_1, \ldots, \beta_p)^T$ and $x = (x_1, \ldots, x_p)^T$.
- It is a $p - 1$ dimensional space
- In the $p = 2$ dimension, a hyperplane is a line.

What is a Hyperplane?

- The mathematical definition of a *hyperplane* is quite simple. In general, a p-dimensional hyperplane is defined by

$$f(x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p = 0,$$

  or equivalently $\beta_0 + \beta^T x = 0$, where $\beta = (\beta_1, \ldots, \beta_p)^T$ and $x = (x_1, \ldots, x_p)^T$.
- It is a $p - 1$ dimensional space
- In the $p = 2$ dimension, a hyperplane is a line.
- For any two points x_a and x_b on the hyperplane, we have $\beta^T(x_a - x_b) = 0$, and hence β is the vector orthogonal to the surface of the hyperplane.

What is a separating hyperplane?

- A *separating hyperplane* is a hyperplane that separates two classes
- All observations with $y_i = +1$ are on one side of the hyperplane and all observations with $y_i = -1$ are on the other side
  - $\beta_0 + \beta_1 x_{1i} + \cdots \beta_p x_{pi} > 0$ when $y_i = 1$
  - $\beta_0 + \beta_1 x_{1i} + \cdots \beta_p x_{pi} < 0$ when $y_i = -1$

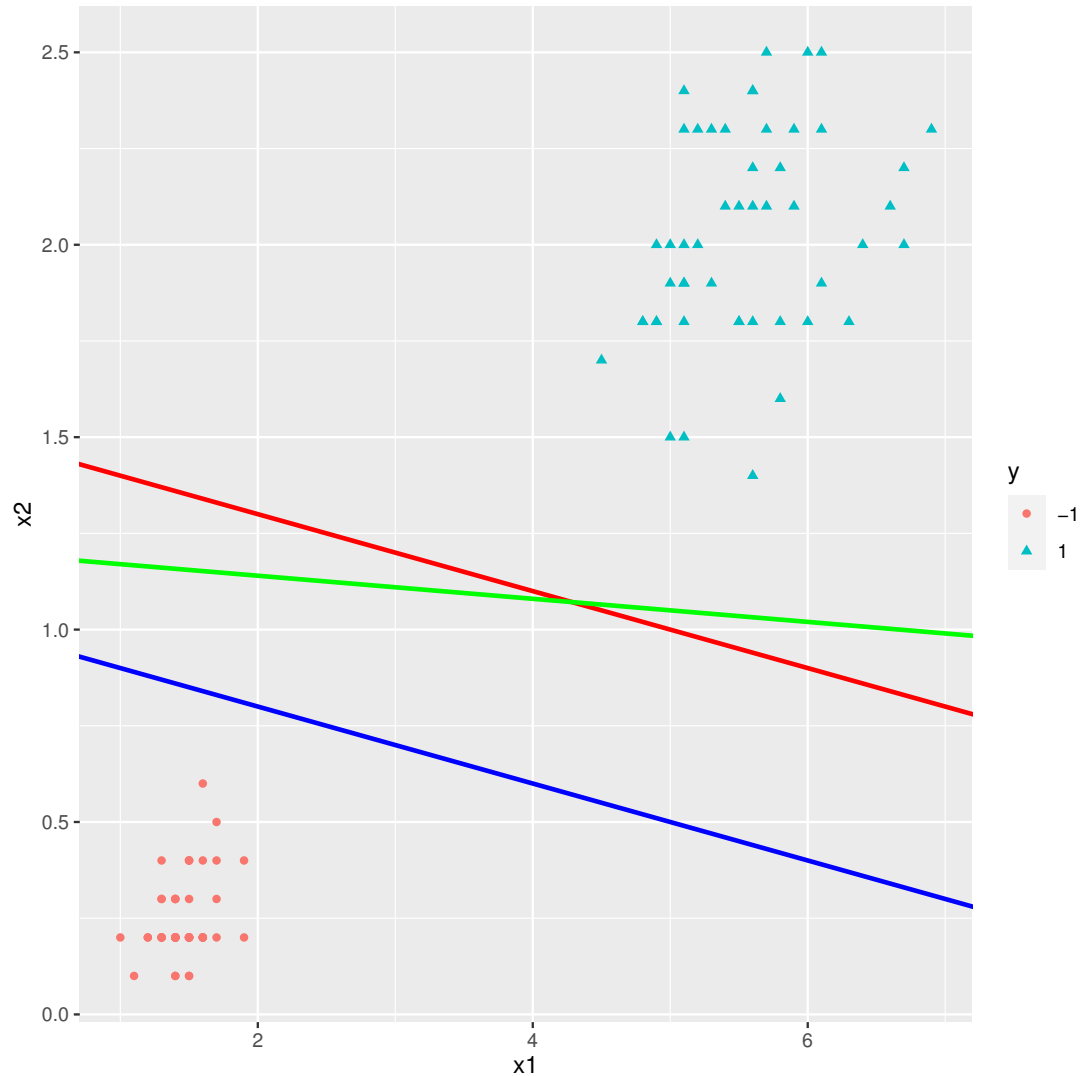
What is a separating hyperplane?

- A *separating hyperplane* is a hyperplane that separates two classes
- All observations with $y_i = +1$ are on one side of the hyperplane and all observations with $y_i = -1$ are on the other side
  - $\beta_0 + \beta_1 x_{1i} + \cdots \beta_p x_{pi} > 0$ when $y_i = 1$
  - $\beta_0 + \beta_1 x_{1i} + \cdots \beta_p x_{pi} < 0$ when $y_i = -1$
- Putting together, let $f(\boldsymbol{x}) = \beta_0 + \beta_1 x_1 + \cdots \beta_p x_p$ we get

$$y_i \times f(\boldsymbol{x}_i) > 0$$

- $f(\boldsymbol{x}) = 0$ defines a *separating hyperplane*.

Some separating hyperplanes for the Iris data

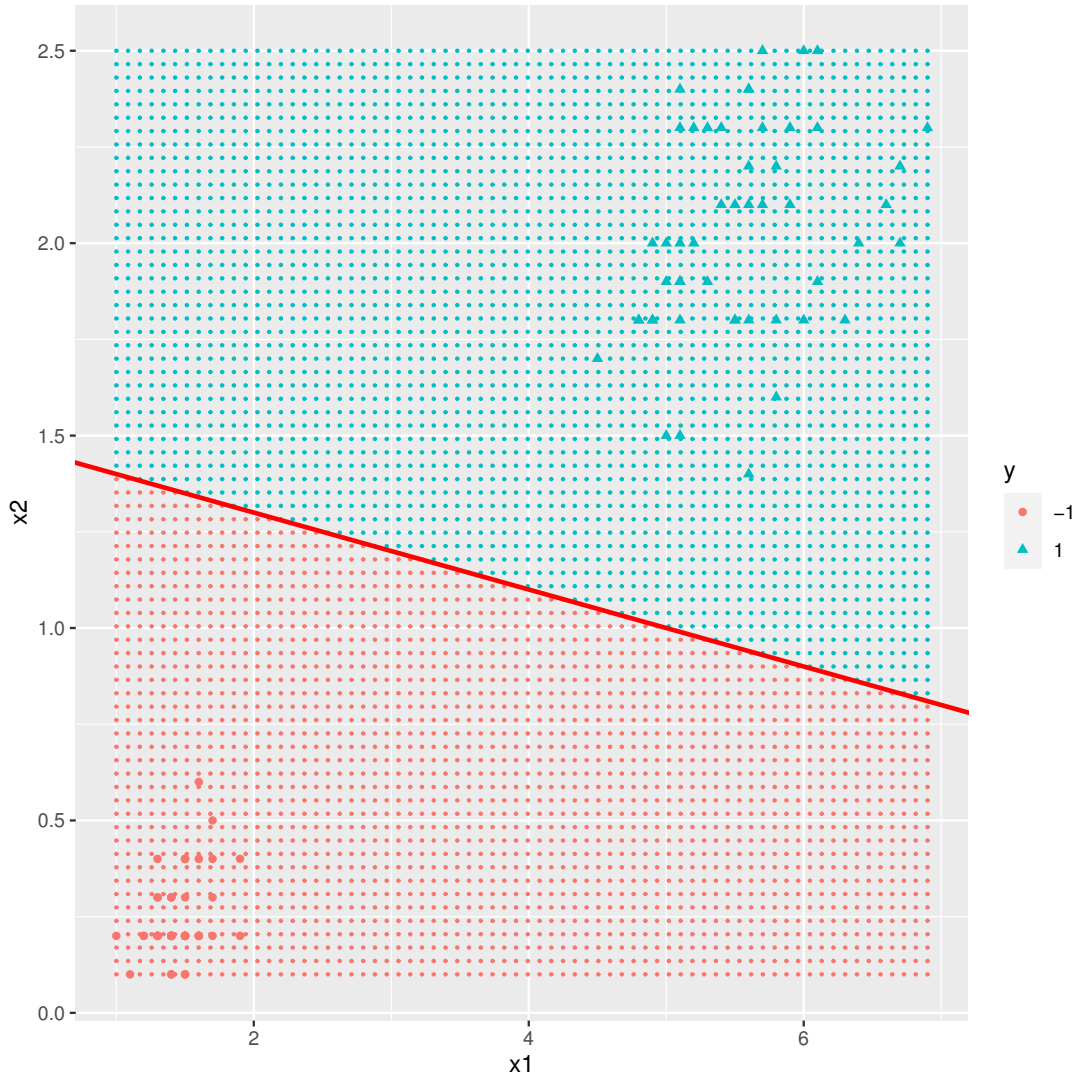


R code for the previous plot

```
## Some separating hyperplanes for the Iris data
ggplot(data=n.iris, aes(x = x1, y = x2)) + geom_point(aes(color=y, shape=y)) +
  xlab("x1") +  ylab("x2") + geom_abline(intercept = 1.5, slope = -0.1,
    color="red", linetype="solid", size=1) + geom_abline(intercept = 1.2, slope = -0.03,
    color="green", linetype="solid", size=1) + geom_abline(intercept = 1.0, slope = -0.1,
    color="blue", linetype="solid", size=1)
```

Linear classifier

- Given separating hyperplane as classifier

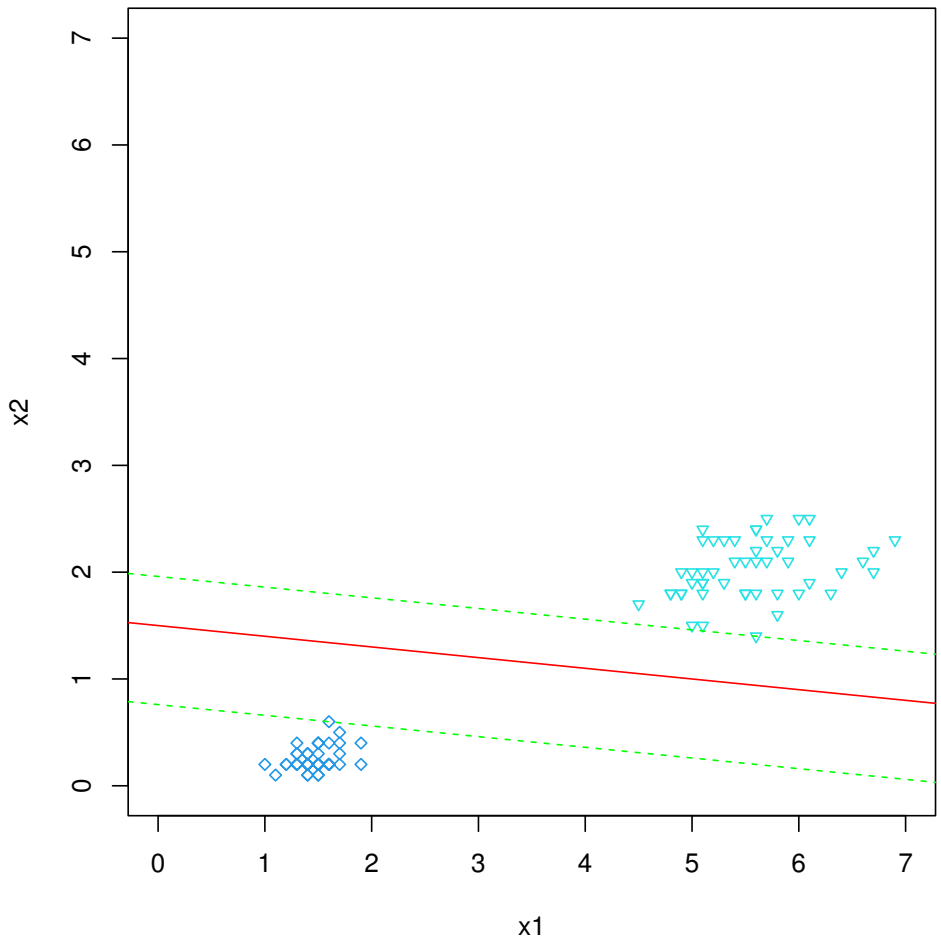


R code for the previous plot

```
## Linear classifier
make.grid = function(x, n = 75) {
  grange = apply(x, 2, range)
  x1 = seq(from = grange[1,1], to = grange[2,1], length = n)
  x2 = seq(from = grange[1,2], to = grange[2,2], length = n)
  expandgrid = expand.grid(x1 = x1, x2 = x2)
  colnames(expandgrid) = colnames(x)
  expandgrid
}
x.grid = make.grid(n.iris[,1:2],n=70)
colnames(x.grid)=c("x1","x2")
x.grid$y = as.factor(ifelse(x.grid$x2 > 1.5 - 0.1*x.grid$x1,1,-1))
ggplot(data=x.grid, aes(x = x1, y = x2)) + geom_point(aes(color=y), size = 0.3) +
  xlab("x1") +  ylab("x2") + geom_abline(intercept = 1.5, slope = -0.1,
    color="red", linetype="solid", size=1) + geom_point(data=n.iris,
      aes(x = x1, y = x2,color=y, shape=y))
```

Maximal margin classifier

- A separating hyperplane comes with "margins"

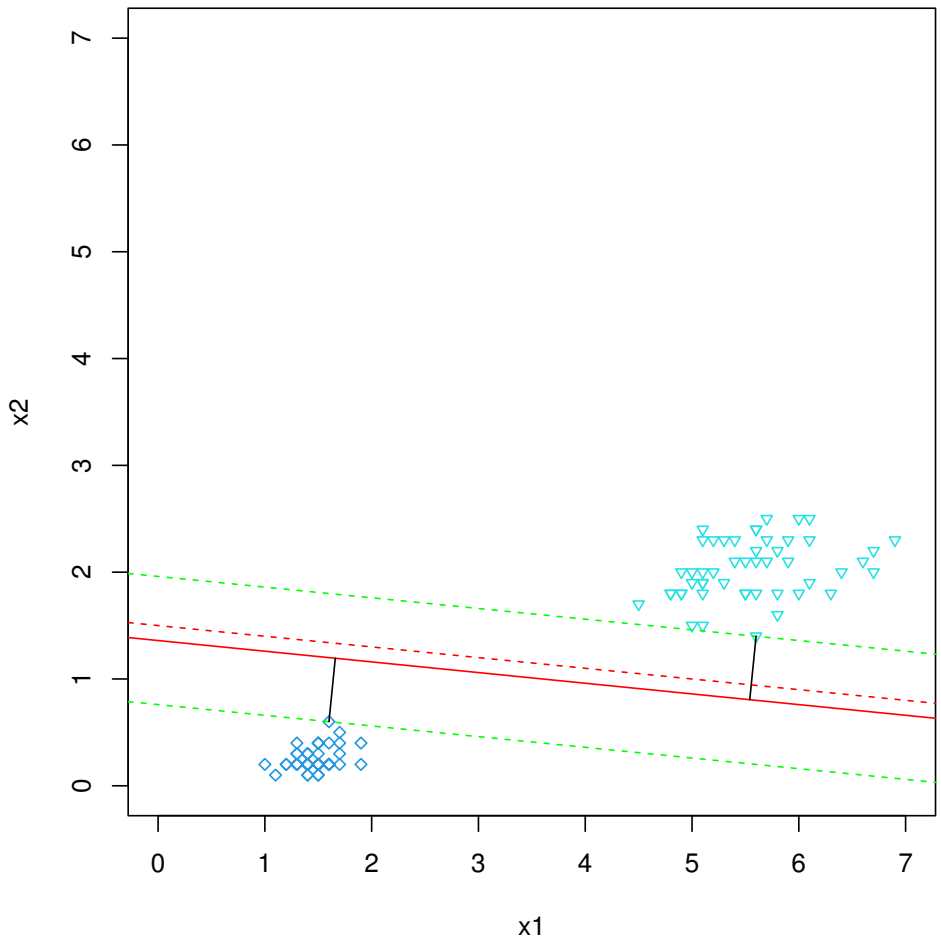


R code for the previous plot

```
## Maximal Margin Classifier
ref.intercept = n.iris$x2 + 0.1*n.iris$x1 - 1.5
iris.margin = c(which.min(ifelse(ref.intercept>0,ref.intercept,100)),
  which.min(-ifelse(ref.intercept<0,ref.intercept,-100)))
b = -0.1
m = 1.5
new.points.x = (n.iris$x2[iris.margin] - m + n.iris$x1[iris.margin]/b)/(b + 1/b)
new.points.y = (m/b+b*n.iris$x2[iris.margin]+n.iris$x1[iris.margin])/(b + 1/b)
old.points.x = n.iris$x1[iris.margin]
old.points.y = n.iris$x2[iris.margin]
par(pty="s")
plot(n.iris$x1,n.iris$x2,xlab = "x1",ylab = "x2",col = as.integer(n.iris$y)+3,pch = as.integer(n.iris$y) + 4, cex = 0.7,
  xlim = c(0,7), ylim = c(0,7))
abline(m,b,col = "red")
abline(ref.intercept[iris.margin[1]]+m,b,lty =2, col = "green")
abline(ref.intercept[iris.margin[2]]+m,b,lty =2, col = "green")
segments(new.points.x,new.points.y,old.points.x,old.points.y,col = "black")
```

Maximal margin classifier

- A separating hyperplane comes with "margins"

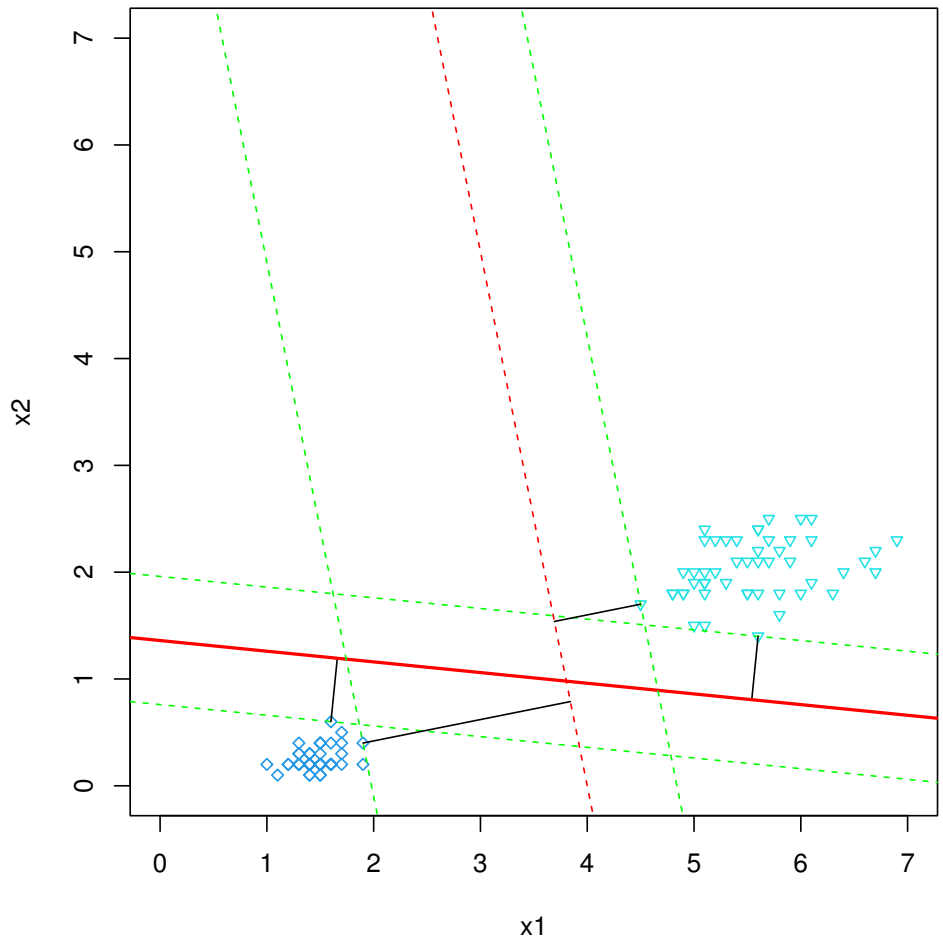


R code for the previous plot

```
## Maximal Margin Classifier
m1 = m
b1 = b
m = mean(ref.intercept[iris.margin])+m
ref.intercept = n.iris$x2 -b*n.iris$x1 - m
iris.margin = c(which.min(ifelse(ref.intercept>0,ref.intercept,100)),
  which.min(-ifelse(ref.intercept<0,ref.intercept,-100)))
new.points.x = (n.iris$x2[iris.margin] - m + n.iris$x1[iris.margin]/b)/(b + 1/b)
new.points.y = (m/b+b*n.iris$x2[iris.margin]+n.iris$x1[iris.margin])/(b + 1/b)
old.points.x = n.iris$x1[iris.margin]
old.points.y = n.iris$x2[iris.margin]
par(pty="s")
plot(n.iris$x1,n.iris$x2,xlab = "x1",ylab = "x2",col = as.integer(n.iris$y)+3,pch = as.integer(n.iris$y) + 4, cex = 0.7,
  xlim = c(0,7), ylim = c(0,7))
abline(m,b,col = "red")
abline(ref.intercept[iris.margin[1]]+m,b,lty =2, col = "green")
abline(ref.intercept[iris.margin[2]]+m,b,lty =2, col = "green")
segments(new.points.x,new.points.y,old.points.x,old.points.y,col = "black")
abline(m1,b,lty = 2,col = "red")
```

Maximal margin classifier

- A separating hyperplane comes with "margins"

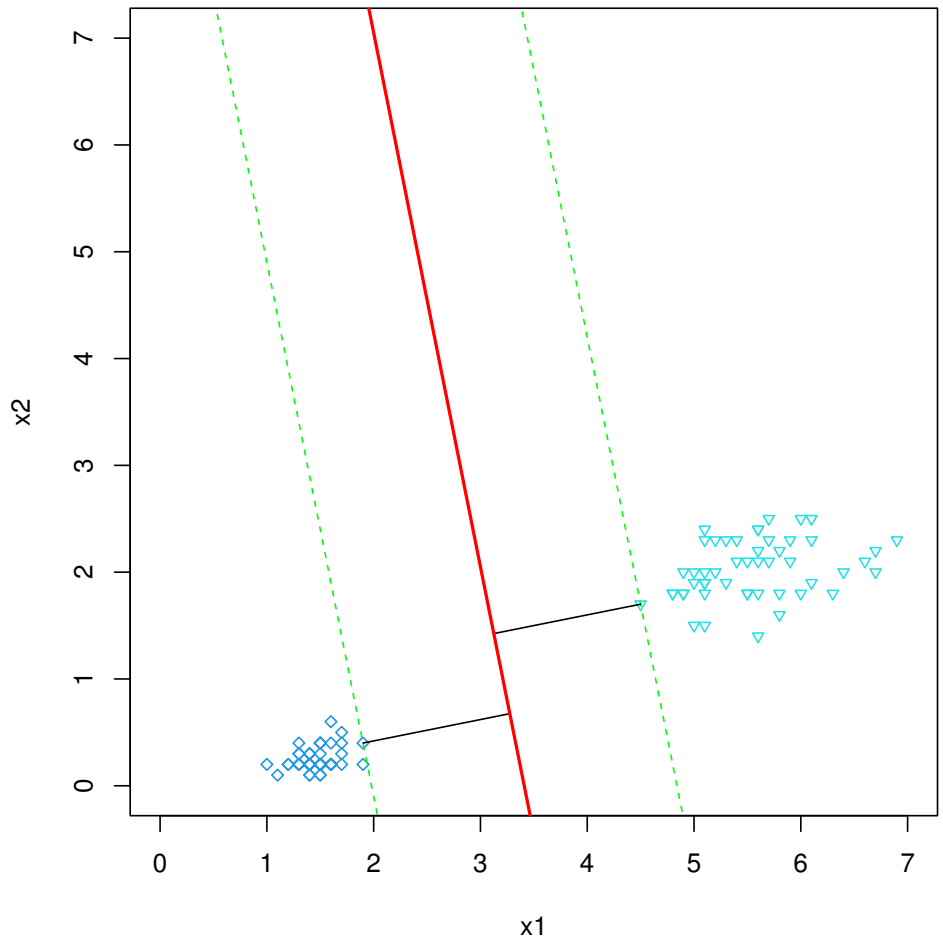


R code for the previous plot

```
## Maximal Margin Classifier
par(pty="s")
plot(n.iris$x1,n.iris$x2,xlab = "x1",ylab = "x2",col = as.integer(n.iris$y)+3,pch = as.integer(n.iris$y) + 4, cex = 0.7,
  xlim = c(0,7), ylim = c(0,7))
abline(m,b,col = "red")
abline(ref.intercept[iris.margin[1]]+m,b,lty =2, col = "green")
abline(ref.intercept[iris.margin[2]]+m,b,lty =2, col = "green")
segments(new.points.x,new.points.y,old.points.x,old.points.y,col = "black")
abline(mean(ref.intercept[iris.margin])+m,b,lwd =2,col = "red")
m = 20
b = -5
abline(m,b,col = "red", lty = 2)
ref.intercept = n.iris$x2 -b*n.iris$x1 - m
iris.margin = c(which.min(ifelse(ref.intercept>0,ref.intercept,100)),
  which.min(-ifelse(ref.intercept<0,ref.intercept,-100)))
new.points.x = (n.iris$x2[iris.margin] - m + n.iris$x1[iris.margin]/b)/(b + 1/b)
new.points.y = (m/b+b*n.iris$x2[iris.margin]+n.iris$x1[iris.margin])/(b + 1/b)
old.points.x = n.iris$x1[iris.margin]
old.points.y = n.iris$x2[iris.margin]
abline(ref.intercept[iris.margin[1]]+m,b,lty =2, col = "green")
abline(ref.intercept[iris.margin[2]]+m,b,lty =2, col = "green")
segments(new.points.x,new.points.y,old.points.x,old.points.y,col = "black")
```

Maximal margin classifier

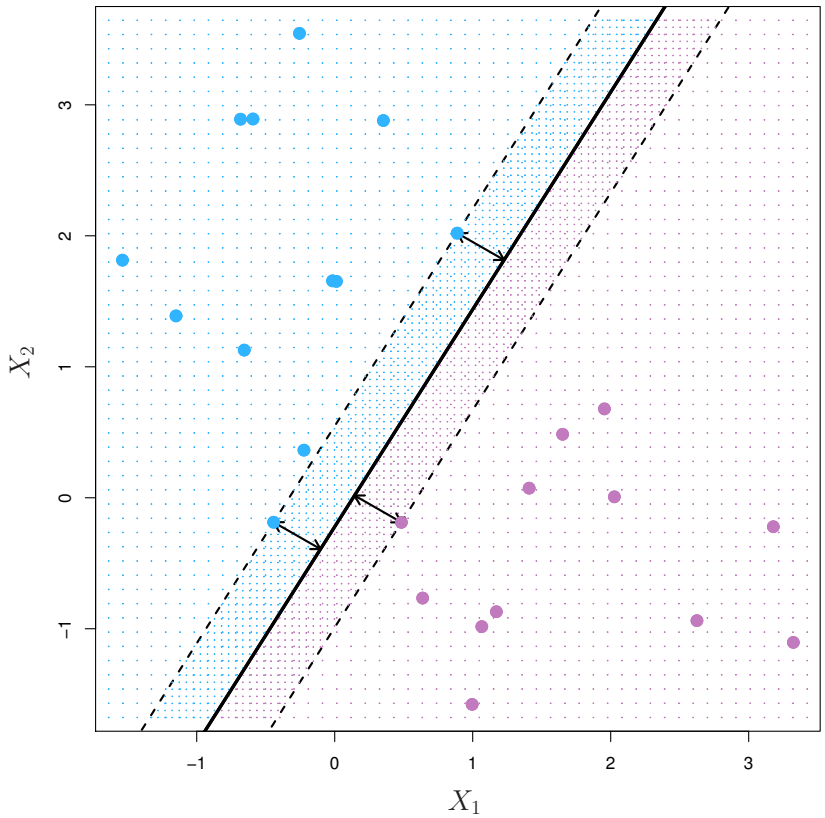
- A separating hyperplane comes with "margins"



R code for the previous plot

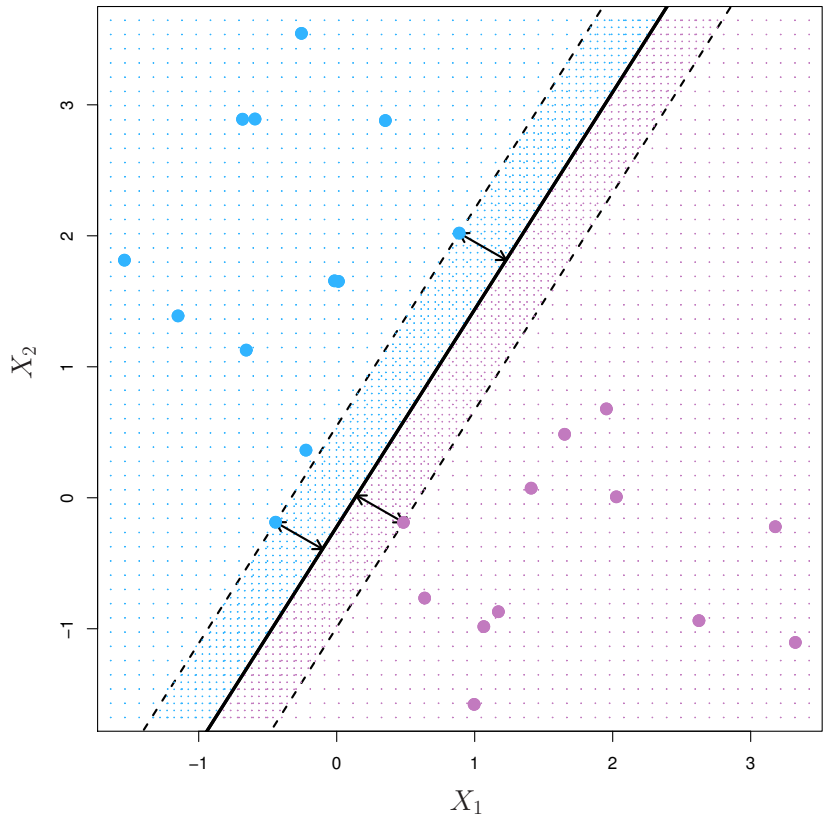
```
## Maximal Margin Classifier
m = mean(ref.intercept[iris.margin])+m
ref.intercept = n.iris$x2 -b*n.iris$x1 - m
iris.margin = c(which.min(ifelse(ref.intercept>0,ref.intercept,100)),
  which.min(-ifelse(ref.intercept<0,ref.intercept,-100)))
new.points.x = (n.iris$x2[iris.margin] - m + n.iris$x1[iris.margin]/b)/(b + 1/b)
new.points.y = (m/b+b*n.iris$x2[iris.margin]+n.iris$x1[iris.margin])/(b + 1/b)
old.points.x = n.iris$x1[iris.margin]
old.points.y = n.iris$x2[iris.margin]
par(pty="s")
plot(n.iris$x1,n.iris$x2,xlab = "x1",ylab = "x2",col = as.integer(n.iris$y)+3,pch = as.integer(n.iris$y) + 4, cex = 0.7,
  xlim = c(0,7), ylim = c(0,7))
abline(m,b,col = "red")
abline(ref.intercept[iris.margin[1]]+m,b,lty =2, col = "green")
abline(ref.intercept[iris.margin[2]]+m,b,lty =2, col = "green")
segments(new.points.x,new.points.y,old.points.x,old.points.y,col = "black")
abline(mean(ref.intercept[iris.margin])+m,b,lwd =2,col = "red")
abline(m,b,col = "red", lty = 2)
```

Maximal margin classifier (textbook)



- Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.

Maximal margin classifier (textbook)



- Among all separating hyperplanes, find the one that makes the biggest gap or margin between the two classes.
- The three observations on the dashed lines are known as *support vectors*, since the maximal margin hyperplane depends directly on them, not one the other observations.

Construction of the Maximal Margin Classifier

- Given a set of n training observations $x_1, \ldots, x_n \in R^p$ and associated class labels $y_1, \ldots, y_n \in \{-1, 1\}$, the maximal margin hyperplane is the solution to

$$\max_{\beta_0, \beta_1, \ldots, \beta_p} M \quad \text{subject to} \tag{1}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1, \tag{2}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M \quad \forall i. \tag{3}$$

- This problem can be more conveniently rephrased as

$$\min_{\beta_0, \beta_1, \ldots, \beta_p} \|\beta\|^2 \quad \text{subject to}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq 1 \quad \forall i.$$

Construction of the Maximal Margin Classifier

- Given a set of n training observations $x_1, \ldots, x_n \in R^p$ and associated class labels $y_1, \ldots, y_n \in \{-1, 1\}$, the maximal margin hyperplane is the solution to

$$\max_{\beta_0, \beta_1, \ldots, \beta_p} M \quad \text{subject to} \tag{1}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1, \tag{2}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M \quad \forall i. \tag{3}$$

- This problem can be more conveniently rephrased as

$$\min_{\beta_0, \beta_1, \ldots, \beta_p} \|\beta\|^2 \quad \text{subject to}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq 1 \quad \forall i.$$

Details of the Maximal Margin Classifier

- The constraint (3) guarantees that each observation will be on the correct side of the hyperplane, provided that M is positive.

Details of the Maximal Margin Classifier

- The constraint (3) guarantees that each observation will be on the correct side of the hyperplane, provided that M is positive.
- Note that (2) is not really a constraint on the hyperplane since if $f(x) = 0$ defines a hyperplane, then so does $kf(x) = 0$ for any $k \neq 0$.
- It turns out that the constrain (2) makes sure that the perpendicular distance from the ith observation to the hyperplane is given by $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})$.

Details of the Maximal Margin Classifier

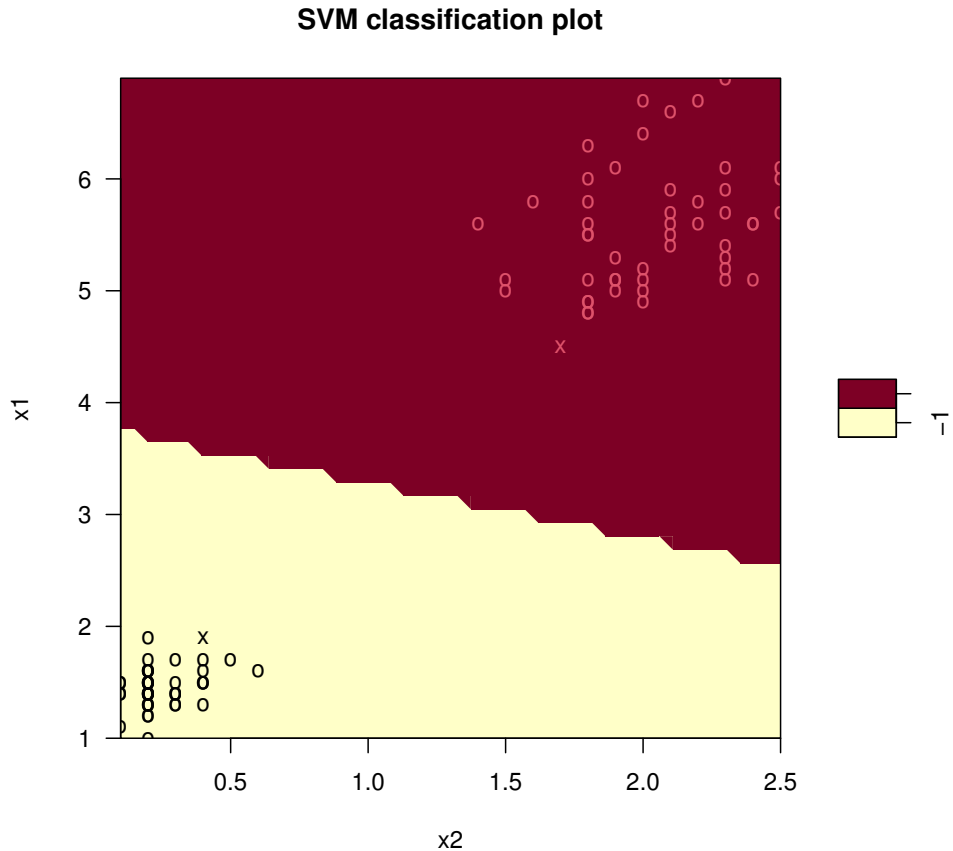
- The constraint (3) guarantees that each observation will be on the correct side of the hyperplane, provided that M is positive.
- Note that (2) is not really a constraint on the hyperplane since if $f(x) = 0$ defines a hyperplane, then so does $kf(x) = 0$ for any $k \neq 0$.
- It turns out that the constrain (2) makes sure that the perpendicular distance from the ith observation to the hyperplane is given by $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})$.
- Therefore, the constraints (2) and (3) ensure that each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane.

Details of the Maximal Margin Classifier

- The constraint (3) guarantees that each observation will be on the correct side of the hyperplane, provided that M is positive.
- Note that (2) is not really a constraint on the hyperplane since if $f(x) = 0$ defines a hyperplane, then so does $kf(x) = 0$ for any $k \neq 0$.
- It turns out that the constrain (2) makes sure that the perpendicular distance from the ith observation to the hyperplane is given by $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})$.
- Therefore, the constraints (2) and (3) ensure that each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane.
- Hence, M represents the margin of our hyperplane, and the optimization problem chooses $\beta_0, \ldots, \beta_p$ to maximize M.

Outline

A quick look at the svm result for Iris data



- "X" in the plot are the support vectors — the points that directly affect the classification line
- "O" are the other points, which don't affect the calculation of the line

R code for the previous plot

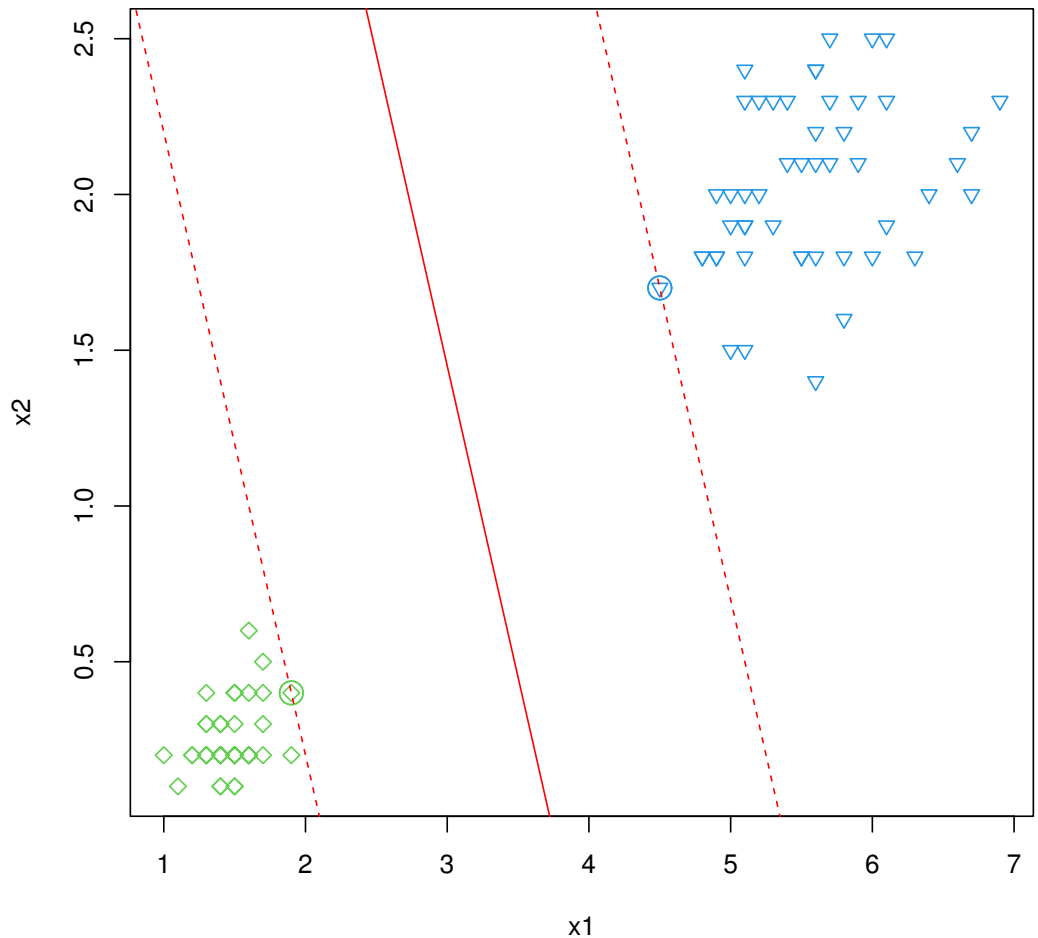
```
> ## Package e1071
> ## A quick look at the svm result for Iris data
> iris.svm1 = svm(y ~ ., data = n.iris, kernel = "linear", scale = F)
> plot(iris.svm1,n.iris)
> iris.svm1$index
[1] 45 57
> iris.svm1$SV
    x1  x2
45  1.9 0.4
107 4.5 1.7
> iris.svm1$rho
[1] -2.292307
> iris.svm1$coefs
         [,1]
[1,]  0.2366864
[2,] -0.2366864
> ?svm
```

Output of the function of svm

An object of class svm containing the fitted model, including:

- SV: The resulting support vectors (possibly scaled).
- index: The index of the resulting support vectors in the data matrix. Note that this index refers to the preprocessed data (after the possible effect of na.omit and subset)
- coefs: The corresponding coefficients times the training labels.
- rho: The negative intercep
- ...

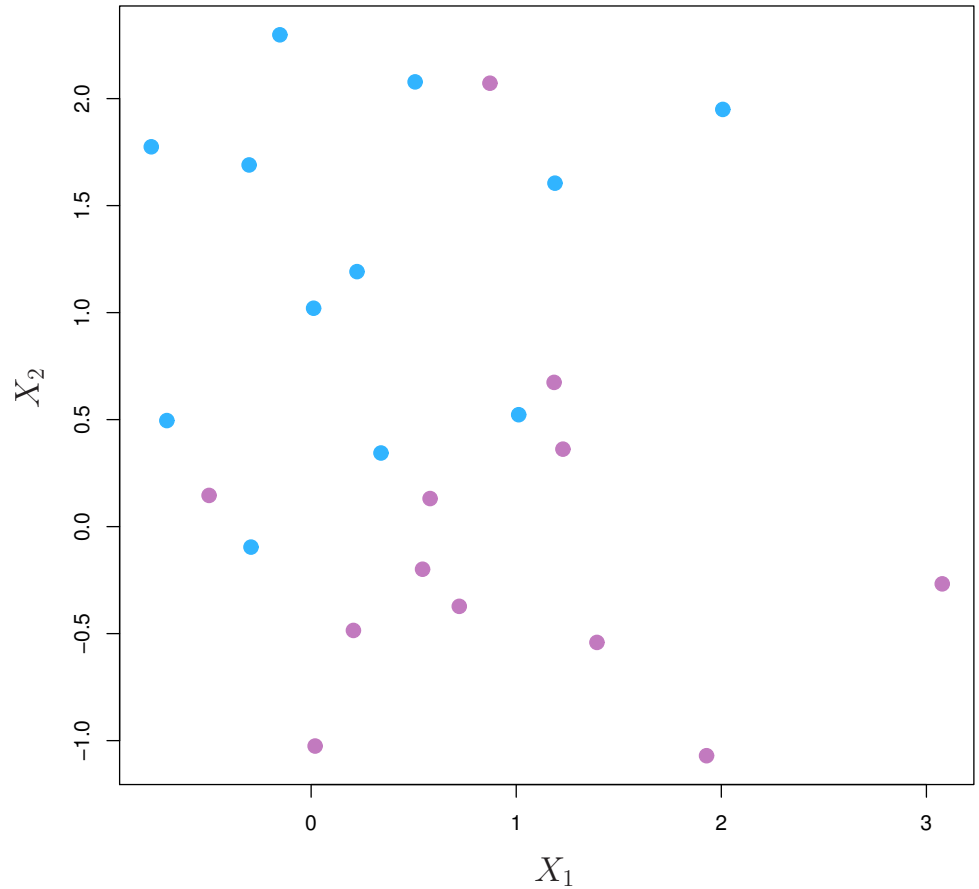
Details in the output



R code for the previous plot

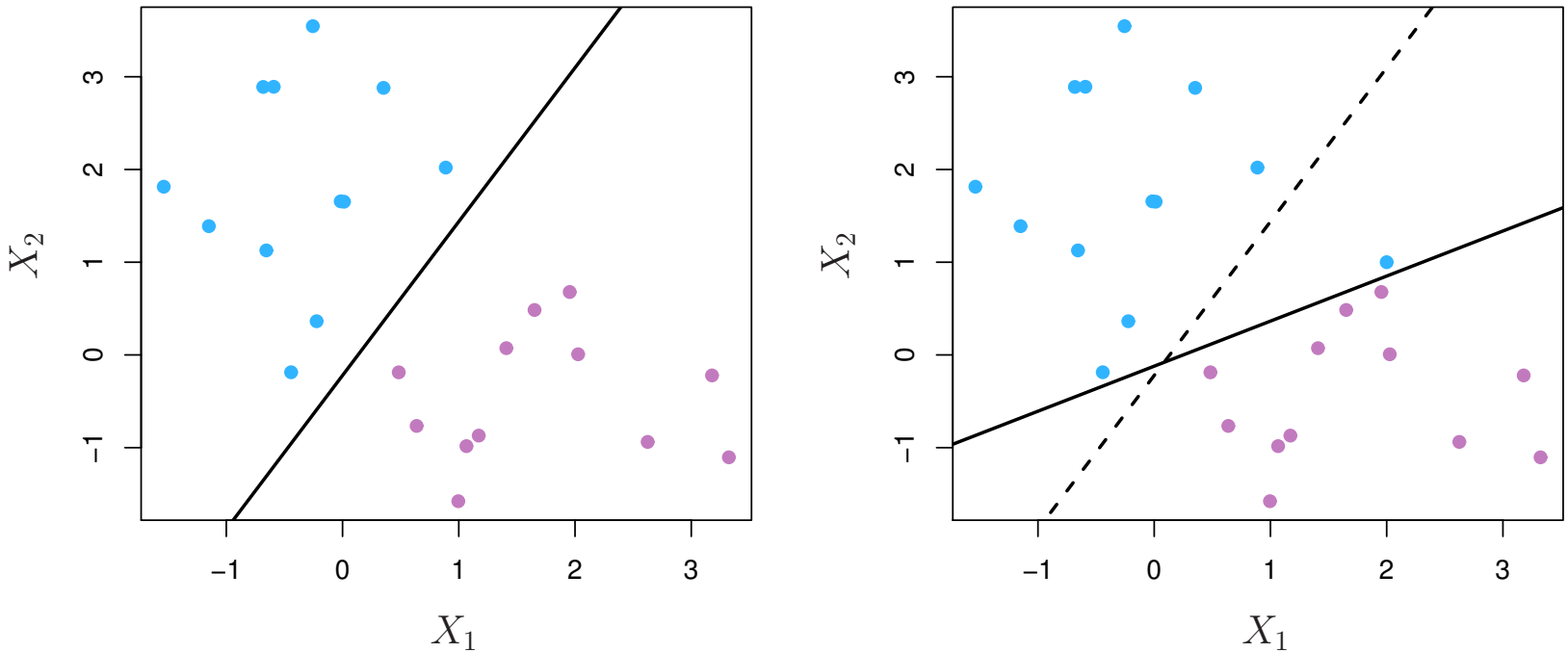
```
## Details in the output
beta = t(iris.svm1$coefs) %*% iris.svm1$SV
beta0 = -iris.svm1$rho
plot(n.iris$x1,n.iris$x2,xlab = "x1",ylab = "x2",col = as.integer(n.iris$y)+2,
  pch = as.integer(n.iris$y) + 4)
# show the support vectors
points(n.iris[iris.svm1$index,c(1,2)],col=as.integer(n.iris[iris.svm1$index,"y"])
  +2, cex=2)
abline(-beta0/beta[1,2],-beta[1,1]/beta[1,2],col = "red")
abline(iris.svm1$SV[1,2] + beta[1,1]/beta[1,2] *iris.svm1$SV[1,1],
      -beta[1,1]/beta[1,2], col = "red",lty = 2)
abline(iris.svm1$SV[2,2] + beta[1,1]/beta[1,2] *iris.svm1$SV[2,1],
      -beta[1,1]/beta[1,2], col = "red",lty = 2)
```

Non-separable Data



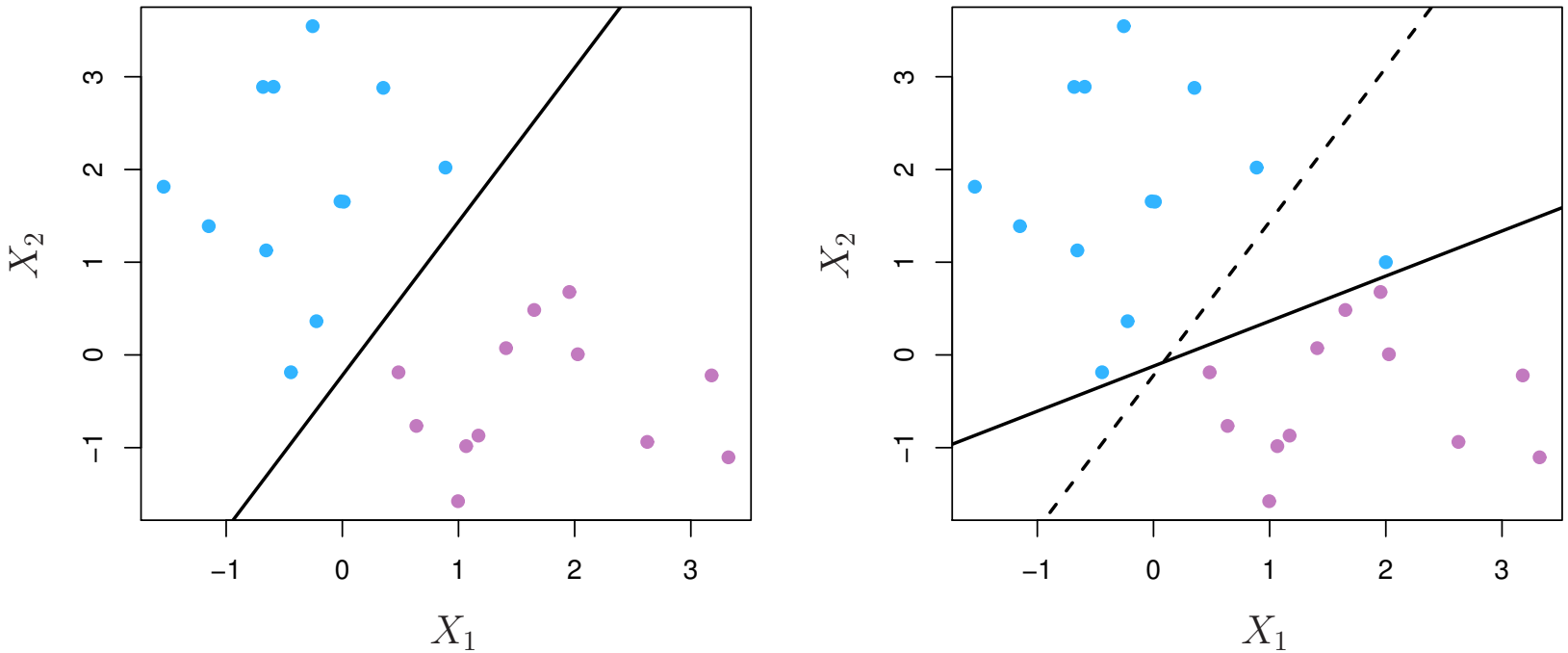
- The two classes are not separable by a hyperplane, and so the maximal margin classifier cannot be used.

Separable but noisy data



- Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.

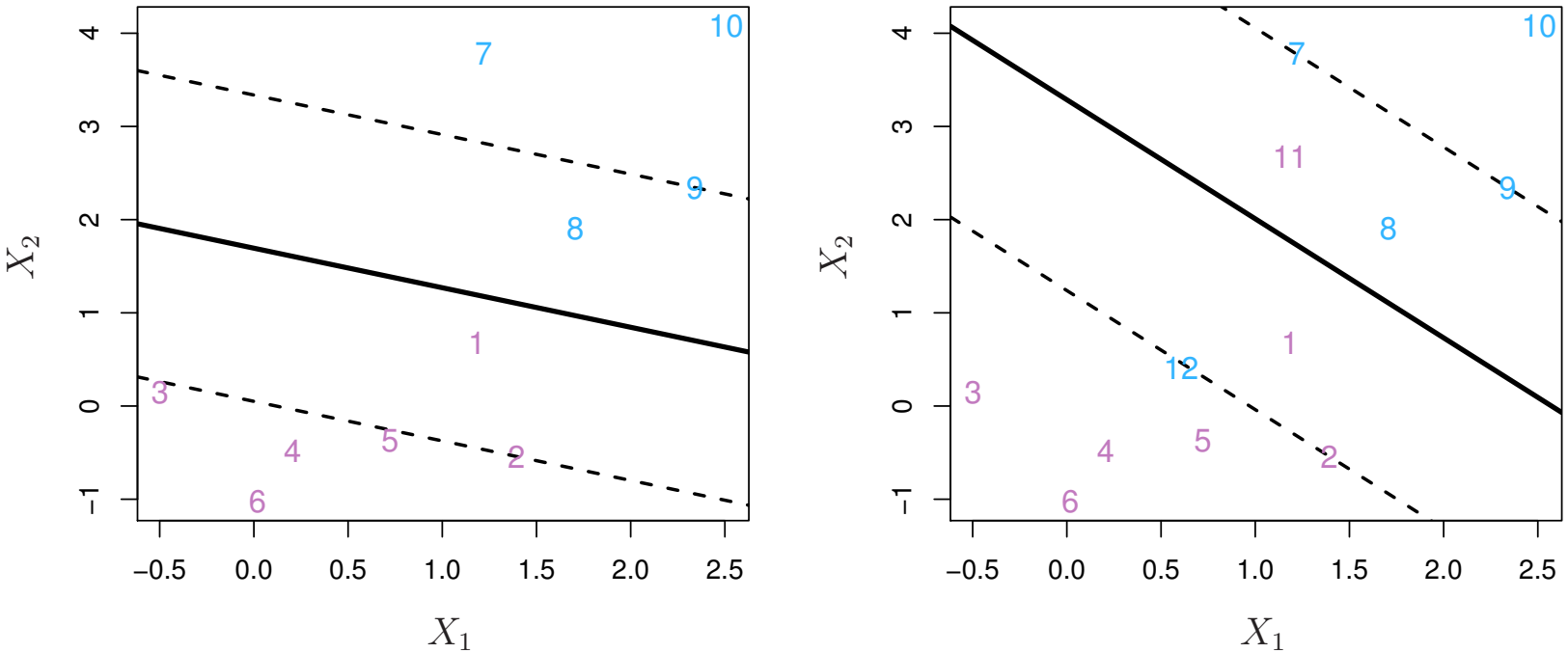
Separable but noisy data



- Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.
- It could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations.

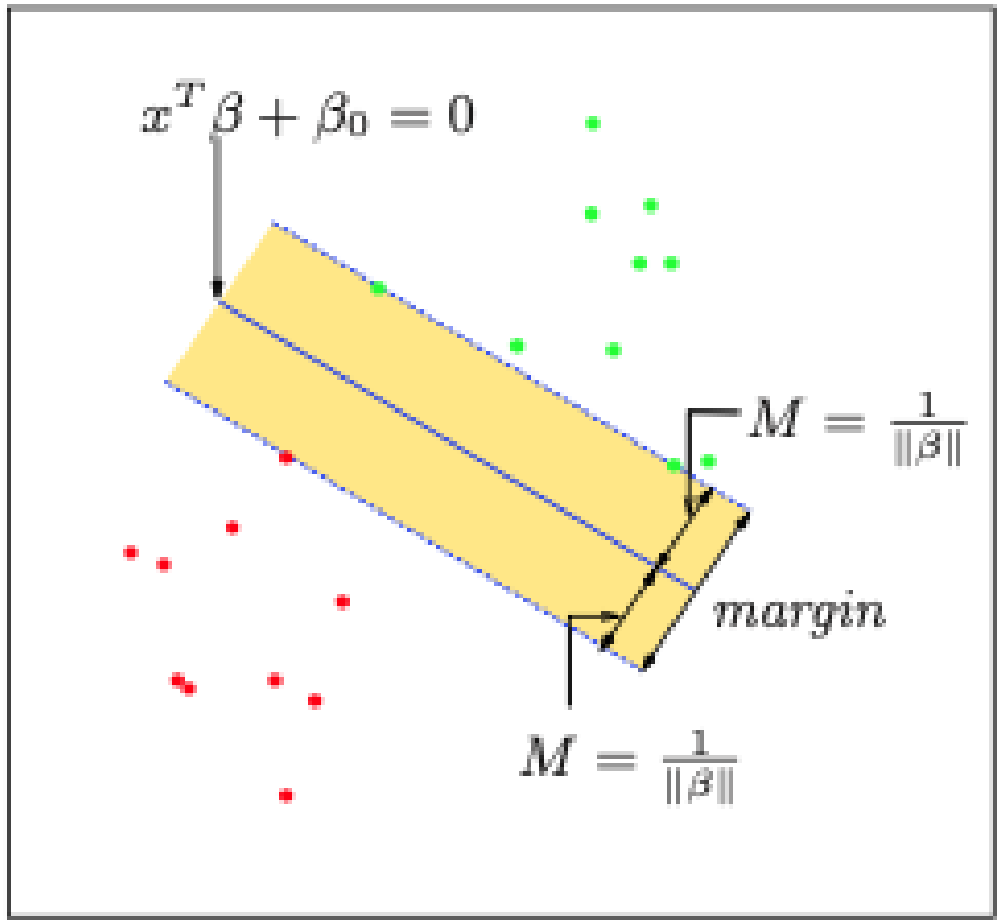
Outline

Support vector classifier



- The support vector classifier maximizes a *soft* margin, which tries to separate most of the training observations into the two classes, but may misclassify a few observations.

Support vector classifier



- The figure above shows the separable case
- The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width $2M = 2/||\beta||$

Optimal separating hyperplanes

- The set of conditions ensure that all the points are at least a signed distance M from the decision boundary defined by $\boldsymbol{\beta}$ and β_0, and we seek the largest such M and associated parameters
  - Actually we can get rid of the $||\boldsymbol{\beta}|| = 1$ constraint by replacing the conditions with

$$\frac{1}{||\boldsymbol{\beta}||} y_i(\boldsymbol{x}_i^T \boldsymbol{\beta} + \beta_0) \geq M \text{ (notice } \beta_0 \text{ redefined)},$$

    or equivalently $y_i(\boldsymbol{x}_i \boldsymbol{\beta} + \beta_0) \geq M||\boldsymbol{\beta}||$
  - Since for any $\boldsymbol{\beta}$ and β_0 satisfying these inequalities, any positively scaled multiple satisfies them too, we can arbitrarily set $||\boldsymbol{\beta}|| = \frac{1}{M}$

Optimal separating hyperplanes

- The set of conditions ensure that all the points are at least a signed distance M from the decision boundary defined by $\boldsymbol{\beta}$ and β_0, and we seek the largest such M and associated parameters
  - Actually we can get rid of the $||\boldsymbol{\beta}|| = 1$ constraint by replacing the conditions with

$$\frac{1}{||\boldsymbol{\beta}||} y_i(\boldsymbol{x}_i^T \boldsymbol{\beta} + \beta_0) \geq M \text{ (notice } \beta_0 \text{ redefined)},$$

    or equivalently $y_i(\boldsymbol{x}_i \boldsymbol{\beta} + \beta_0) \geq M||\boldsymbol{\beta}||$
  - Since for any $\boldsymbol{\beta}$ and β_0 satisfying these inequalities, any positively scaled multiple satisfies them too, we can arbitrarily set $||\boldsymbol{\beta}|| = \frac{1}{M}$
  - The optimization problem is equivalent to

$$\min_{\boldsymbol{\beta}, \beta_0} \frac{1}{2} ||\boldsymbol{\beta}||^2, \text{ subject to } y_i(\boldsymbol{x}_i^T \boldsymbol{\beta} + \beta_0) \geq 1, i = 1, \ldots, n$$

Optimal separating hyperplanes

- The set of conditions ensure that all the points are at least a signed distance M from the decision boundary defined by $\boldsymbol{\beta}$ and β_0, and we seek the largest such M and associated parameters
  - Actually we can get rid of the $||\boldsymbol{\beta}|| = 1$ constraint by replacing the conditions with

$$\frac{1}{||\boldsymbol{\beta}||} y_i(\boldsymbol{x}_i^T \boldsymbol{\beta} + \beta_0) \geq M \text{ (notice } \beta_0 \text{ redefined)},$$

    or equivalently $y_i(\boldsymbol{x}_i \boldsymbol{\beta} + \beta_0) \geq M||\boldsymbol{\beta}||$
  - Since for any $\boldsymbol{\beta}$ and β_0 satisfying these inequalities, any positively scaled multiple satisfies them too, we can arbitrarily set $||\boldsymbol{\beta}|| = \frac{1}{M}$
  - The optimization problem is equivalent to

$$\min_{\boldsymbol{\beta}, \beta_0} \frac{1}{2}||\boldsymbol{\beta}||^2, \text{ subject to } y_i(\boldsymbol{x}_i^T \boldsymbol{\beta} + \beta_0) \geq 1, i = 1, \ldots, n$$

- This is a convex optimization problem (quadratic criterion with linear inequality constraints)

Lagrange method

- The Lagrange function, to be minimized w.r.t. $\boldsymbol{\beta}$ and β_0, is

$$L_P = \frac{1}{2}||\boldsymbol{\beta}||^2 - \sum_{i=1}^{n} \alpha_i [y_i(\boldsymbol{x}_i^T \boldsymbol{\beta}) - 1]$$

Lagrange method

- The Lagrange function, to be minimized w.r.t. $\boldsymbol{\beta}$ and β_0, is

$$L_P = \frac{1}{2}||\boldsymbol{\beta}||^2 - \sum_{i=1}^{n} \alpha_i [y_i(\boldsymbol{x}_i^T \boldsymbol{\beta}) - 1]$$

- Setting partial derivatives to zeroes, we obtain

$$\boldsymbol{\beta} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i, \text{ and } 0 = \sum_{i=1}^{n} \alpha_i y_i$$

- Substituting these in L_P we obtain the so-called Wolfe dual

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{n} \alpha_i \alpha_k y_i y_k \boldsymbol{x}_i^T \boldsymbol{x}_k$$

subject to $\alpha_i \geq 0$, for $i = 1, \ldots, n$. This is a simple convex optimization problem, for which standard software can be used

An illustration example of Lagrange multiplier

- Suppose you are running a factory, producing some sort of widget that requires steel as a raw material
  - our costs are predominantly human labor, which is \$20 per hour for your workers
  - The steel itself, which runs for \$170 per ton

An illustration example of Lagrange multiplier

- Suppose you are running a factory, producing some sort of widget that requires steel as a raw material
  - our costs are predominantly human labor, which is \$20 per hour for your workers
  - The steel itself, which runs for \$170 per ton
- Suppose your revenue R is loosely modeled by the equation $R(h, s) = 200h^{2/3}s^{1/3}$
  - h: hours of labor
  - s: tons of steel

An illustration example of Lagrange multiplier

- Suppose you are running a factory, producing some sort of widget that requires steel as a raw material
  - our costs are predominantly human labor, which is \$20 per hour for your workers
  - The steel itself, which runs for \$170 per ton
- Suppose your revenue R is loosely modeled by the equation $R(h, s) = 200h^{2/3}s^{1/3}$
  - h: hours of labor
  - s: tons of steel
- If your budget is $\$20,000$, what is the maximum possible revenue?

An illustration example of Lagrange multiplier

- The restriction is $20h + 170s = 20,000$

An illustration example of Lagrange multiplier

- The restriction is $20h + 170s = 20,000$
- Lagrange function: $L(h, s, \lambda) = \ln(R) - \lambda \times \text{restriction} = \ln(200) - \frac{2}{3}\ln(h) + \frac{1}{3}\ln(s) - \lambda(20h + 170s - 20000)$

An illustration example of Lagrange multiplier

- The restriction is $20h + 170s = 20,000$
- Lagrange function: $L(h, s, \lambda) = \ln(R) - \lambda \times \text{restriction} = \ln(200) - \frac{2}{3}\ln(h) + \frac{1}{3}\ln(s) - \lambda(20h + 170s - 20000)$
- Taking partial derivatives and setting them as zeroes:
  - $0 = -\frac{2}{3h} - 20\lambda \implies 20h = \frac{2}{3\lambda}$

An illustration example of Lagrange multiplier

- The restriction is $20h + 170s = 20,000$
- Lagrange function: $L(h, s, \lambda) = \ln(R) - \lambda \times \text{restriction} = \ln(200) - \frac{2}{3}\ln(h) + \frac{1}{3}\ln(s) - \lambda(20h + 170s - 20000)$
- Taking partial derivatives and setting them as zeroes:
  - $0 = -\frac{2}{3h} - 20\lambda \implies 20h = \frac{2}{3\lambda}$
  - $0 = -\frac{1}{3s} - 170\lambda \implies 170s = \frac{1}{3\lambda}$
  - Since $20h + 170s = 20000$, we obtain $\lambda = \frac{1}{20000}$
    - Hence, $\hat{h} = \frac{1}{30\lambda} = \frac{20000}{30} = 667$, and $\hat{s} = \frac{1}{3 \cdot 170\lambda} = \frac{20000}{510} = 39$
    - The maximum revenue would be $\hat{R}^{max} = 200\hat{h}^{2/3}\hat{s}^{1/3} = \$51,854.82$

Supporting points

- Solutions of the convex optimization problem satisfy conditions on previous page, and

$$\alpha_i[y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) - 1] = 0, \ \forall i$$

  - If $\alpha_i > 0$, then $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) = 1$, or in other words, $\boldsymbol{x}_i$ is on the boundary of the slab
  - If $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) > 1$, $\boldsymbol{x}_i$ is not on the boundary of the slab, and $\alpha_i = 0$

Supporting points

- Solutions of the convex optimization problem satisfy conditions on previous page, and

$$\alpha_i[y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) - 1] = 0, \ \forall i$$

  - If $\alpha_i > 0$, then $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) = 1$, or in other words, $\boldsymbol{x}_i$ is on the boundary of the slab
  - If $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) > 1$, $\boldsymbol{x}_i$ is not on the boundary of the slab, and $\alpha_i = 0$
- Solution of vector $\boldsymbol{\beta}$ is defined in terms of a linear combination of the *support points* $\boldsymbol{x}_i$ — those points defined to be on the boundary of the slab via $\alpha_i > 0$
  - β_0 can be solved for any support points

Supporting points

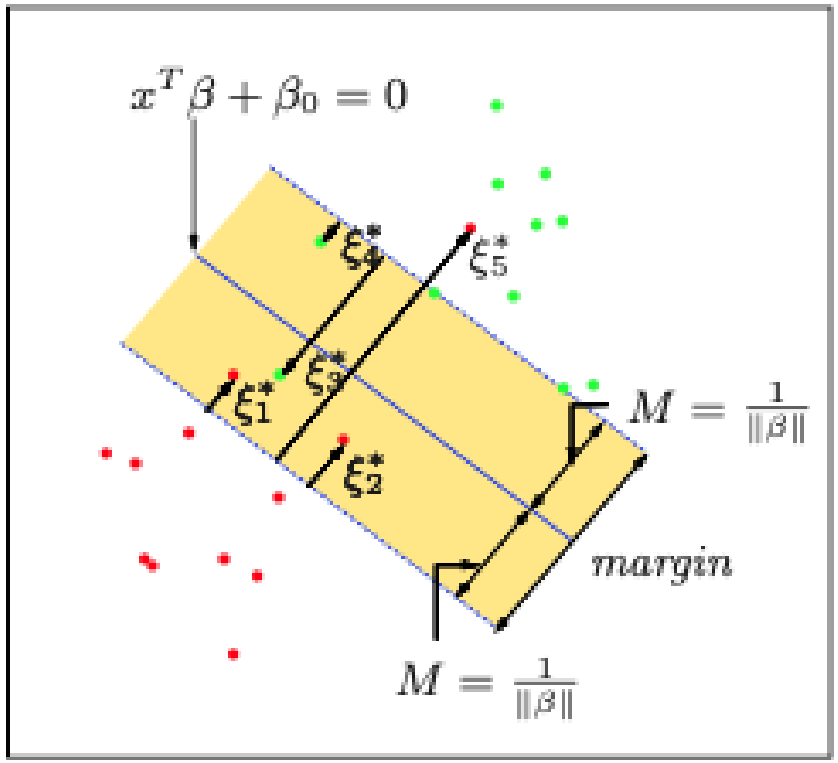
- Solutions of the convex optimization problem satisfy conditions on previous page, and

$$\alpha_i[y_i(\boldsymbol{x}_i^T\boldsymbol{\beta}+\beta_0)-1]=0,\ \forall i$$

  - If $\alpha_i > 0$, then $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta}+\beta_0)=1$, or in other words, $\boldsymbol{x}_i$ is on the boundary of the slab
  - If $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta}+\beta_0)>1$, $\boldsymbol{x}_i$ is not on the boundary of the slab, and $\alpha_i = 0$
- Solution of vector $\boldsymbol{\beta}$ is defined in terms of a linear combination of the *support points* $\boldsymbol{x}_i$ — those points defined to be on the boundary of the slab via $\alpha_i > 0$
  - β_0 can be solved for any support points
- The optimal separating hyperplane produces a function $\hat{f}(\boldsymbol{x})=\boldsymbol{x}^T\hat{\boldsymbol{\beta}}+\hat{\beta}_0$ for classifying new observations: $\hat{G}(\boldsymbol{x})=\text{sign}\hat{f}(\boldsymbol{x})$

Support vector classifier

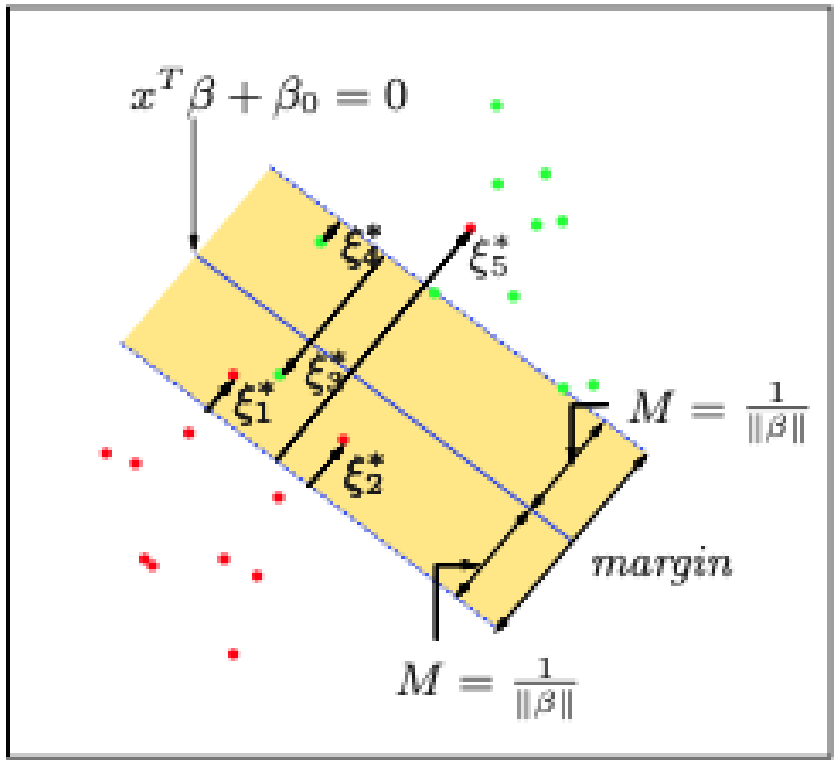
- As mentioned above, the classes are not clearly separable.



- The points labeled ξ_j^* are on the wrong side of their margin by an amount $\xi_j^* = M\xi_j$, and points on the correct side have $\xi_j^* = 0$

Support vector classifier

- As mentioned above, the classes are not clearly separable.



- The points labeled ξ_j^* are on the wrong side of their margin by an amount $\xi_j^* = M\xi_j$, and points on the correct side have $\xi_j^* = 0$
- The margin is maximized subject to a total budget $\sum \xi_i \leq C$, a constant, i.e., $\sum \xi_j^*$ is the total distance of points on the wrong side of their margin

SVM details for overlap classes

- Define the slack variables $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_n)$
  - There are two natural ways to modify the constraint $y_i(\boldsymbol{x}_i^T \boldsymbol{\beta} + \beta_0) \geq M$

SVM details for overlap classes

- Define the slack variables $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_n)$
  - There are two natural ways to modify the constraint $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) \geq M$
  - $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) \geq M - \xi_i$, $\forall i, \xi_i \geq 0, \sum_{i=1}^n \xi_i \leq C$, or

SVM details for overlap classes

- Define the slack variables $\boldsymbol{\xi} = (\xi_1, \xi_2, \ldots, \xi_n)$
  - There are two natural ways to modify the constraint $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) \geq M$
  - $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) \geq M - \xi_i$, $\forall i, \xi_i \geq 0, \sum_{i=1}^n \xi_i \leq C$, or
  - $y_i(\boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0) \geq M(1 - \xi_i)$, $\forall i, \xi_i \geq 0, \sum_{i=1}^n \xi_i \leq C$
  - The first choice seems more natural, since it measures overlap in actual distance from the margin; but it results in a nonconvex optimization problem
  - The second choice measures the overlap in relative distance, which changes with the width of the margin M; it is convex
- The second choice leads to the "standard" support vector classifier, which is used often

Construction of the Support Vector Classifier

- The hyperplane for the support vector classifier is the solution to the optimization problem

$$\max_{\beta_0,\beta_1,\ldots,\beta_p,\xi_1,\ldots,\xi_n} M \quad \text{subject to}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M(1 - \xi_i) \quad \forall i,$$

$$\xi_i \geq 0, \ \sum_{i=1}^{n} \xi_i \leq C,$$

where C is a nonnegative tuning parameter.

Construction of the Support Vector Classifier

- The hyperplane for the support vector classifier is the solution to the optimization problem

$$\max_{\beta_0,\beta_1,\ldots,\beta_p,\xi_1,\ldots,\xi_n} M \quad \text{subject to}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M(1 - \xi_i) \quad \forall i,$$

$$\xi_i \geq 0, \ \sum_{i=1}^{n} \xi_i \leq C,$$

  where C is a nonnegative tuning parameter.

- M is the width of the margin; we seek to make this quantity as large as possible.

Construction of the Support Vector Classifier

- The hyperplane for the support vector classifier is the solution to the optimization problem

$$\max_{\beta_0,\beta_1,\ldots,\beta_p,\xi_1,\ldots,\xi_n} M \quad \text{subject to}$$

$$\sum_{j=1}^{p} \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq M(1 - \xi_i) \quad \forall i,$$

$$\xi_i \geq 0, \ \sum_{i=1}^{n} \xi_i \leq C,$$

  where C is a nonnegative tuning parameter.

- M is the width of the margin; we seek to make this quantity as large as possible.
- ξ_is are *slack variables* that allow individual observations to be on the wrong side of the margin or the hyperplane.

Details of the support vector classifier

- The slack variable ξ_i tells us where the ith observation is located, relative to the hyperplane and relative to the margin.
  - If $\xi_i = 0$ then the ith observation is on the correct side of the margin.
  - If $\xi_i > 0$ then the ith observation is on the wrong side of the margin.
  - If $\xi_i > 1$ then the ith observation is on the wrong side of the hyperplane.

Details of the support vector classifier

- The slack variable ξ_i tells us where the ith observation is located, relative to the hyperplane and relative to the margin.
  - If $\xi_i = 0$ then the ith observation is on the correct side of the margin.
  - If $\xi_i > 0$ then the ith observation is on the wrong side of the margin.
  - If $\xi_i > 1$ then the ith observation is on the wrong side of the hyperplane.
- C determines the number and severity of the violations to the margin (and to the hyperplane) that we will tolerate. For $C > 0$, no more than C observations can be on the wrong side of the hyperplane. Why?

Cost: C is a regularization parameter

- For the clearly separable data

```
> iris.svm2 = svm(y ~ ., data = n.iris, kernel = "linear", cost =1)
> iris.svm2$SV
        x1       x2
44  -0.9101666 -0.5838055
107  0.4739357  0.6143028
> iris.svm3 = svm(y ~ ., data = n.iris, kernel = "linear", cost = 10000)
> iris.svm3$SV
        x1       x2
44  -0.9101666 -0.5838055
107  0.4739357  0.6143028
```

- It seems that both "Cost" yield same results

```
> iris.svm4 = svm(y ~ ., data = n.iris, kernel = "linear", cost =0.5)
> iris.svm4$SV
        x1       x2
44  -0.9101666 -0.5838055
45  -0.7669836 -0.8016434
107  0.4739357  0.6143028
120  0.7125740  0.3964649
```

- This time, we see more support vectors.

“Hard” margin versus “Soft” margin

- When we don’t allow any samples to be inside the margin, we are talking about the **hard** margin

“Hard” margin versus “Soft” margin

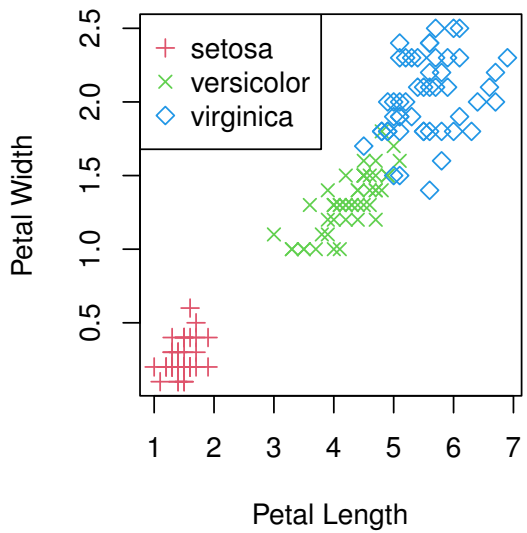
- When we don’t allow any samples to be inside the margin, we are talking about the **hard** margin
- More often, we use **soft** margin classification, in which we allow certain samples to be inside the margin
  - By doing so, the overall fit of the model to the data might very well be better than with hard margin classification
  - It would reduce variance at the cost of some bias (i.e., the bias-variance trade-off)

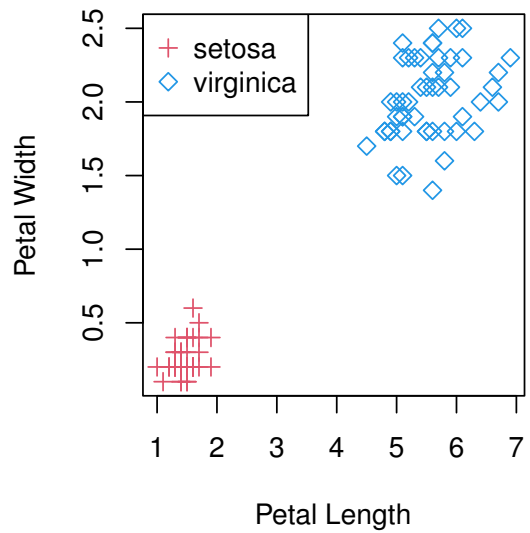
"Hard" margin versus "Soft" margin

- When we don't allow any samples to be inside the margin, we are talking about the **hard** margin
- More often, we use **soft** margin classification, in which we allow certain samples to be inside the margin
  - By doing so, the overall fit of the model to the data might very well be better than with hard margin classification
  - It would reduce variance at the cost of some bias (i.e., the bias-variance trade-off)
- So, here is where the "Cost" is introduced
  - Purpose: modify the optimization problem to optimize both the fit of the line to data and penalizing the amount of samples inside the margin at the same time
  - C defines the weight of how much samples inside the margin contribute to the overall error: the low value of C allows more samples inside the margin; near 0 means every sample can be inside

The iris data we used

- Look at the original iris data and the iris data we classed earlier



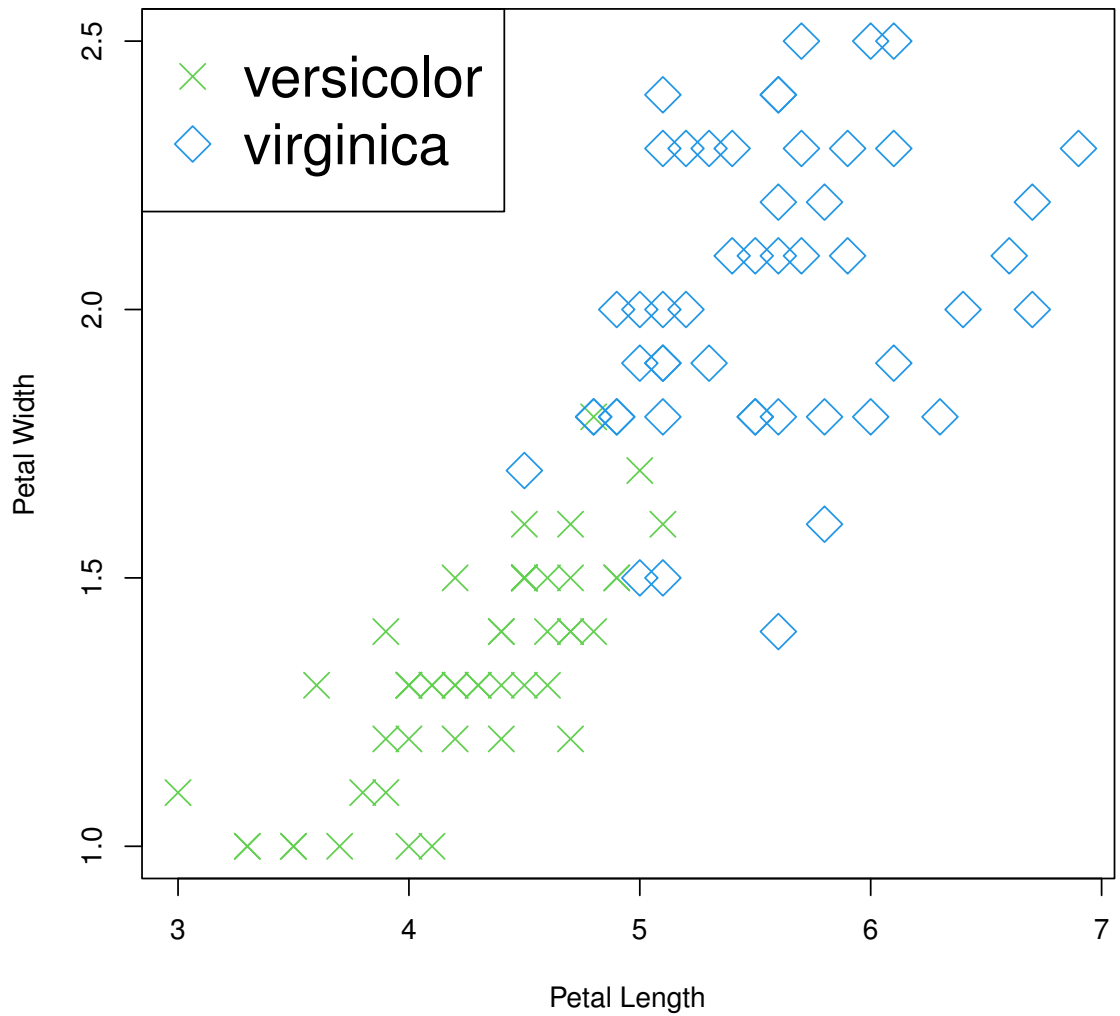


R code for the previous plot

```
## The iris data we used
par(mfrow = c(1,2))
par(pty = "s")
plot(iris$Petal.Length,iris$Petal.Width,col = as.numeric(iris$Species)+1,
  pch = as.numeric(iris$Species)+2, xlab = "Petal Length",ylab = "Petal Width")
legend("topleft",legend = unique(iris$Species), col =
as.numeric(unique(iris$Species))+1,
  pch = as.numeric(unique(iris$Species))+2)
par(pty = "s")
plot(n.iris$x1,n.iris$x2,col = 2*as.numeric(n.iris$y),xlab = "Petal Length",
  ylab = "Petal Width",pch = 2*as.numeric(n.iris$y)+1)
legend("topleft",legend = c("setosa","virginica"), col = c(2,4),pch = c(3,5))
```

The "other" iris data we used

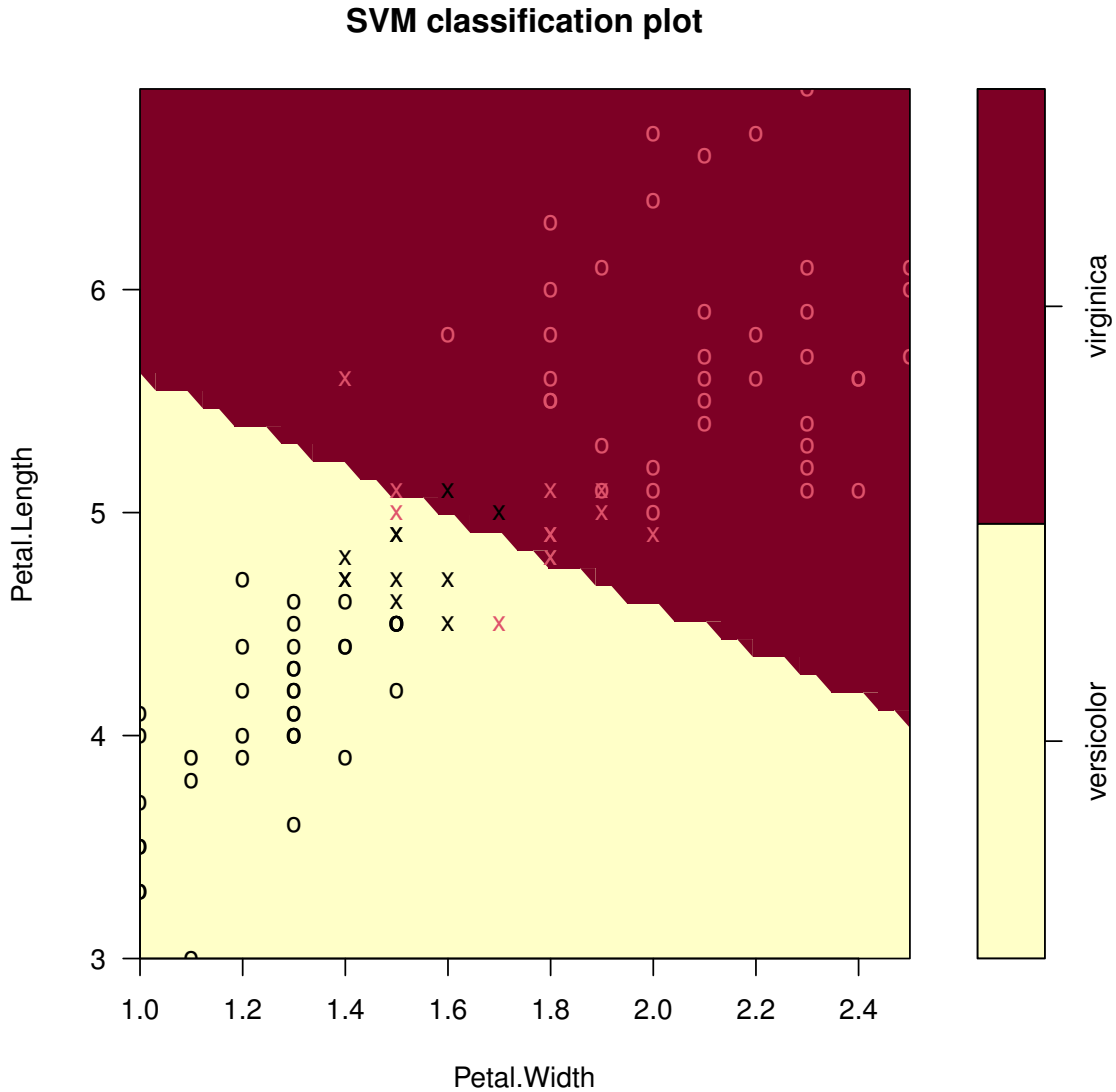
- Now let's consider only "versicolor" and "virginica", which are not clearly classed.



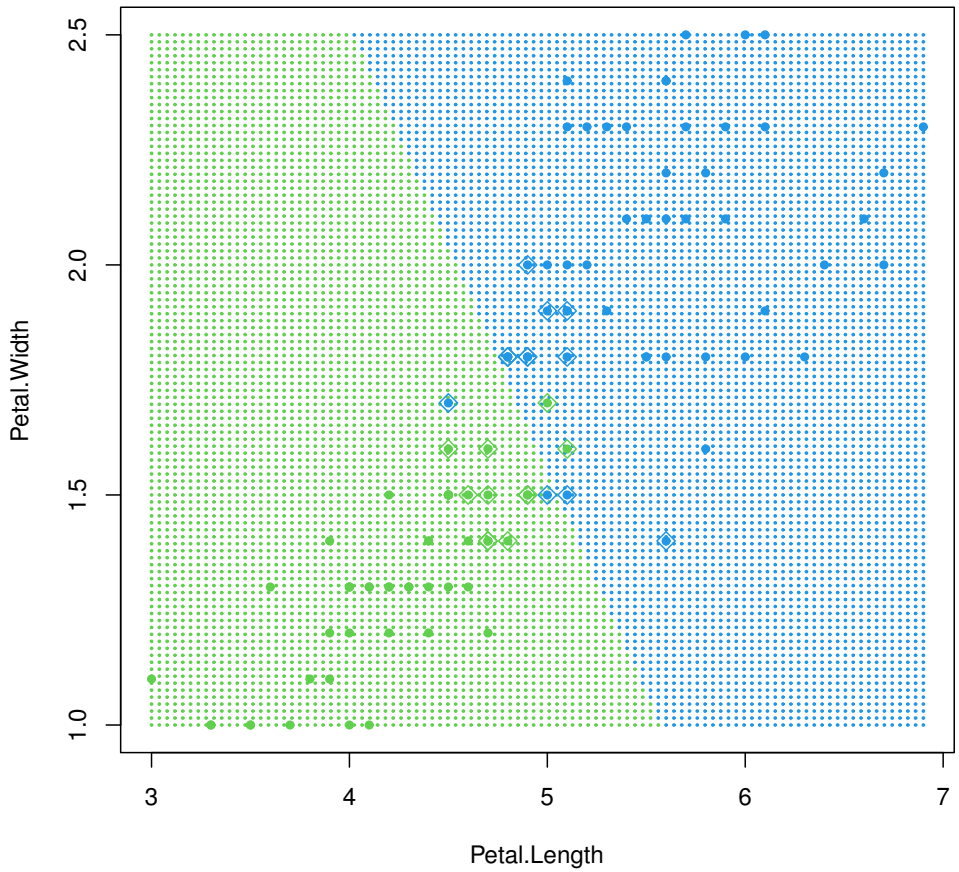
R code for the previous plot

```
## The "other" iris data
iris2 = iris[iris$Species != "setosa",c("Petal.Length","Petal.Width","Species")]
iris2$Species=as.factor(as.character(iris2$Species))
plot(iris2$Petal.Length,iris2$Petal.Width,col = as.numeric(iris2$Species)+2,
  pch = as.numeric(iris2$Species)+3, xlab = "Petal Length",ylab = "Petal Width",cex =2)
legend("topleft",legend = unique(iris2$Species), col =
as.numeric(unique(iris2$Species))+2,
  pch = as.numeric(unique(iris2$Species))+3,cex = 2)

## SVM classifier
iris2.svm1 = svm(Species ~ ., data = iris2, kernel = "linear",scale = F)
plot(iris2.svm1,iris2,cex = 2) #next page for figure
```

SVM classifier



Support points

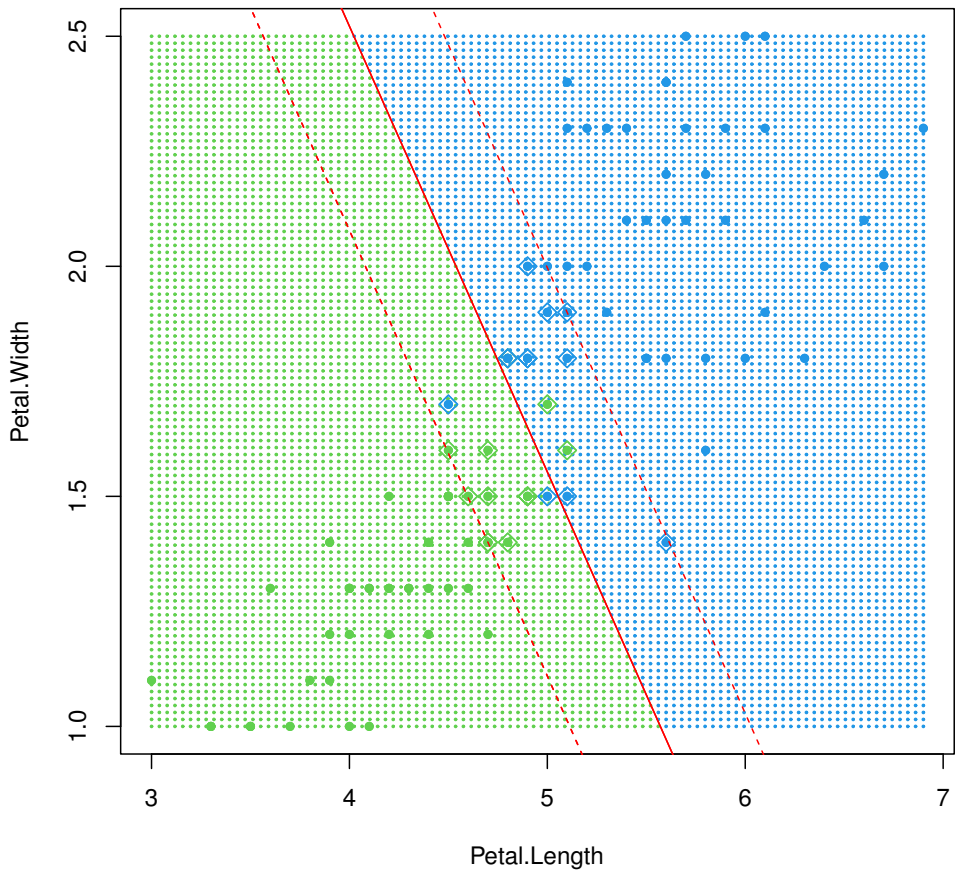


- The support points in the boxes are close to the decision boundary and are instrumental in determining that boundary.

R code for the previous plot

```
## Support Points
x.grid = make.grid(iris2[,1:2], n = 100)
y.grid = predict(iris2.svm1, x.grid)
plot(x.grid,col = as.numeric(y.grid)+2,pch = 20,cex=0.3)
points(iris2[,1:2],col = as.numeric(iris2$Species)+2, pch = 19, cex = 0.7)
points(iris2[iris2.svm1$index,1:2], pch = 5, cex = 1.2, col =
as.numeric(iris2[iris2.svm1$index,3])+2)
```

Support vectors

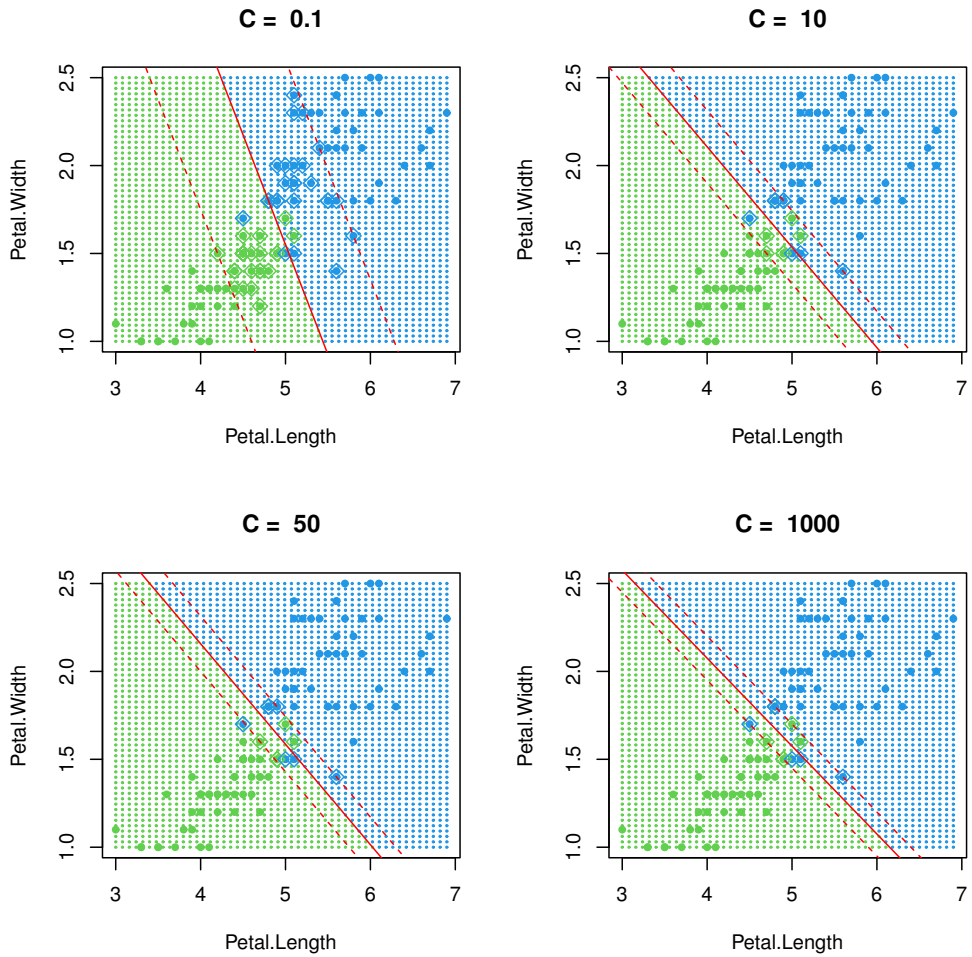


R code for the previous plot

```
## Support vectors
plot(x.grid,col = as.numeric(y.grid)+2,pch = 20,cex=0.3)
points(iris2[,1:2],col = as.numeric(iris2$Species)+2, pch = 19, cex = 0.7)
points(iris2[iris2.svm1$index,1:2], pch = 5, cex = 1.2, col =
as.numeric(iris2[iris2.svm1$index,3])+2)

beta = t(iris2.svm1$coefs) %*% iris2.svm1$SV
beta0 = -iris2.svm1$rho
abline(-beta0/beta[1,2],-beta[1,1]/beta[1,2],col = "red")
abline(-(beta0-1)/beta[1,2],-beta[1,1]/beta[1,2],col = "red",lty = 2)
abline(-(beta0+1)/beta[1,2],-beta[1,1]/beta[1,2],col = "red",lty = 2)
```

Variation of cost: C

- The default cost in svm() is 1



R code for the previous plot

```
## Variation of Cost: $C$
par(mfrow = c(2,2))
x.grid = make.grid(iris2[,1:2], n = 50)
for(c in c(0.1,10,50,1000)){
iris2.svm1 = svm(Species ~ ., data = iris2, kernel = "linear",scale = F, cost = c)
y.grid = predict(iris2.svm1, x.grid)
plot(x.grid,col = as.numeric(y.grid)+2,pch = 20,cex=0.3, main = paste("C = ",c))
points(iris2[,1:2],col = as.numeric(iris2$Species)+2, pch = 19, cex = 0.7)
points(iris2[iris2.svm1$index,1:2], pch = 5, cex = 1.2,
  col = as.numeric(iris2[iris2.svm1$index,3])+2)

beta = t(iris2.svm1$coefs) %*% iris2.svm1$SV
beta0 = -iris2.svm1$rho
abline(-beta0/beta[1,2],-beta[1,1]/beta[1,2],col = "red")
abline(-(beta0-1)/beta[1,2],-beta[1,1]/beta[1,2],col = "red",lty = 2)
abline(-(beta0+1)/beta[1,2],-beta[1,1]/beta[1,2],col = "red",lty = 2)
}
```

Use cross-validation for tuning of C

```
> set.seed(1)
> iris2.tune = tune(svm, Species ~ ., data = iris2, kernel = "linear", scale = F,
+   ranges = list(cost = c(0.001,0.01,0.1,0.5,1,5,10,50,100)))
> summary(iris2.tune)
Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
 cost
  0.1

- best performance: 0.05

- Detailed performance results:
   cost error dispersion
1 1e-03  0.40 0.30550505
2 1e-02  0.07 0.08232726
3 1e-01  0.05 0.08498366
4 5e-01  0.06 0.06992059
5 1e+00  0.05 0.07071068
6 5e+00  0.06 0.09660918
7 1e+01  0.07 0.10593499
8 5e+01  0.07 0.10593499
9 1e+02  0.07 0.10593499
```

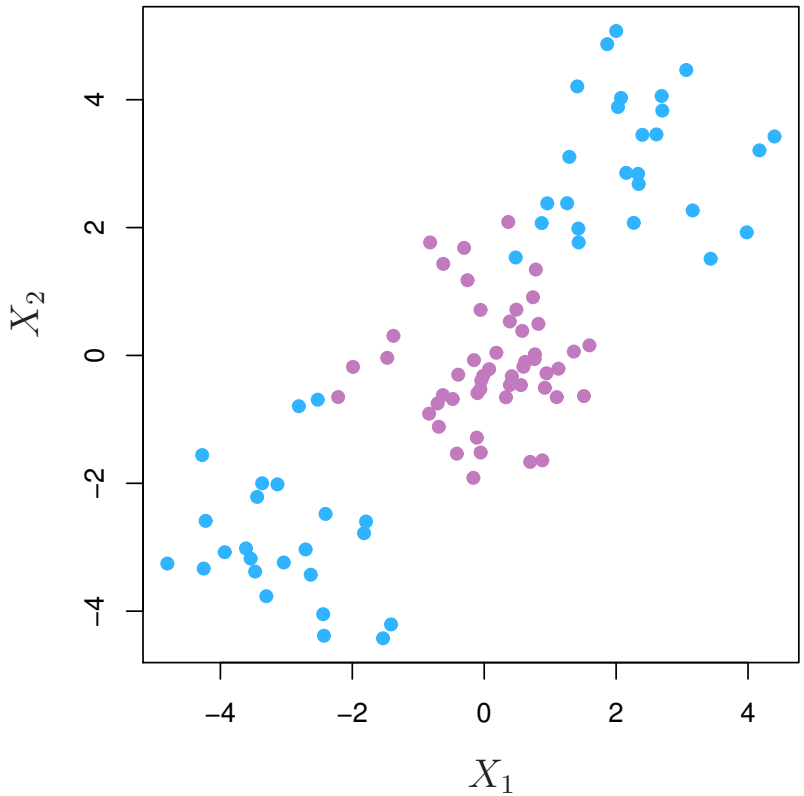
Fine tuning of C

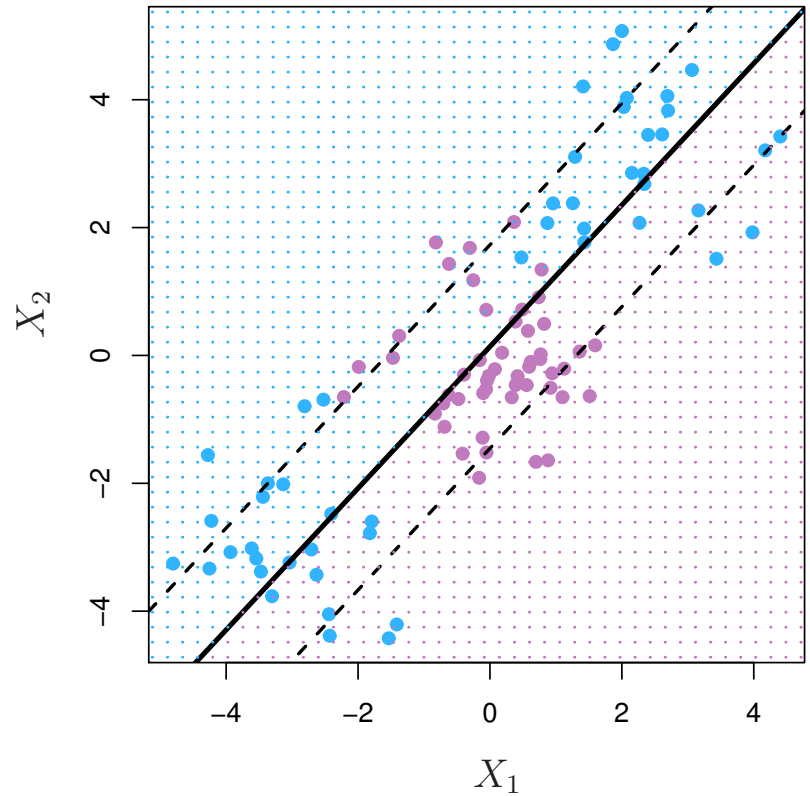
```
> set.seed(1)
> iris2.tune = tune(svm, Species ~ ., data = iris2, kernel = "linear", scale = F,
+   ranges = list(cost = seq(0.02,0.2,0.02)))
> summary(iris2.tune)
Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
 cost
 0.06

- best performance: 0.05

- Detailed performance results:
   cost error dispersion
1  0.02  0.06 0.06992059
2  0.04  0.06 0.08432740
3  0.06  0.05 0.08498366
4  0.08  0.05 0.08498366
5  0.10  0.05 0.08498366
6  0.12  0.05 0.08498366
7  0.14  0.06 0.08432740
8  0.16  0.06 0.08432740
9  0.18  0.06 0.08432740
10 0.20  0.06 0.08432740
```

Fine tuning of C

```
> ## Tuning of $C$ (Don't be superstitious)
> iris2.tune = tune(svm, Species ~ ., data = iris2, kernel = "linear", scale = F,
+   ranges = list(cost = seq(0.02,0.2,0.02)))
> summary(iris2.tune)
Parameter tuning of ‘svm’:
- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.02

- best performance: 0.06
- Detailed performance results:
   cost error dispersion
1  0.02  0.06 0.06992059
2  0.04  0.06 0.06992059
3  0.06  0.07 0.08232726
4  0.08  0.07 0.08232726
5  0.10  0.08 0.07888106
6  0.12  0.08 0.07888106
7  0.14  0.07 0.08232726
8  0.16  0.07 0.08232726
9  0.18  0.08 0.07888106
10 0.20  0.08 0.07888106
```

Linear boundary can fail



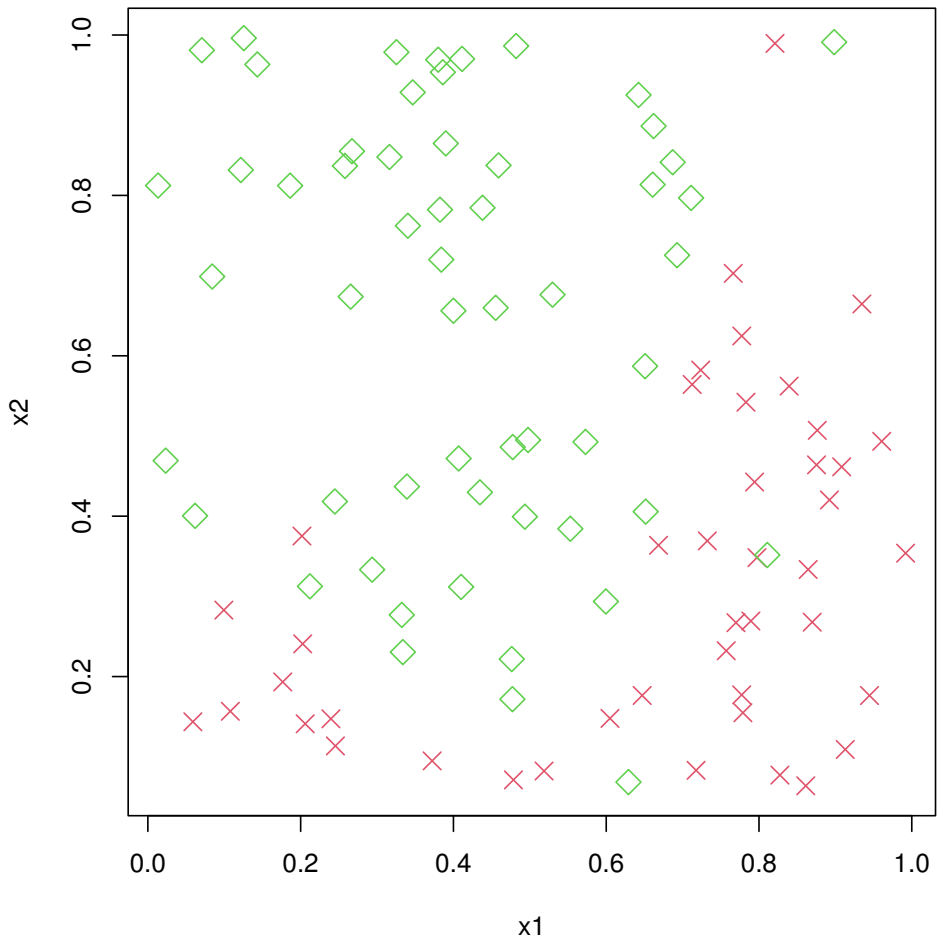


- In practice we are sometimes faced with non-linear class boundaries.
- It is clear that a support vector classifier or any linear classifier will perform poorly here.

Outline

A simulated data

- We have a training data set in $[0,1]^2$ with 100 observations and two classes



R codes for the previous figure

```
## A simulated data
y.fun = function(x,alpha){
  (alpha*((2*x)^2*exp(-1+2*x +0.3*(2*x)^2)-2*x)+(1-alpha)*(0.5-(2*x)^2/2-
0.8*(2*x)^3)+0.5)/2
}
m = 300
alpha = 0.52
beta = 0.05
set.seed(1)
x = runif(2*m)
z = rbinom(m,1,beta)
df = data.frame(x1=x[1:m],x2=x[(m+1):(2*m)], z= z)
df$y = ifelse(df$x2 > y.fun(df$x1,alpha),1,0)
df$y = as.factor(ifelse(df$z==1,1-df$y,df$y))
df$z = NULL

tr.ind = 1:100
df.tr = df[tr.ind,]
df.te = df[-tr.ind,]
par(pty = "s")
plot(df.tr$x1,df.tr$x2, xlab = "x1",ylab = "x2", col = as.numeric(df.tr$y)+1,
  pch = as.numeric(df.tr$y)+3,cex = 1.5)
```

Linear SVM with tuning

```
> C = c(0.01,0.05,seq(0.1,1,0.1),2,5,10)
> sim.tune = tune(svm, y ~ ., data = df.tr, kernel = "linear", scale = F,
+   ranges = list(cost = C))
> summary(sim.tune)
Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
 cost
  0.4
- best performance: 0.17
- Detailed performance results:
   cost error dispersion
1   0.01  0.44  0.2065591
2   0.05  0.44  0.2065591
3   0.10  0.38  0.2299758
4   0.20  0.18  0.1751190
5   0.30  0.18  0.1751190
6   0.40  0.17  0.1766981
7   0.50  0.17  0.1888562
8   0.60  0.21  0.1728840
9   0.70  0.20  0.1563472
10  0.80  0.19  0.1728840
11  0.90  0.18  0.1686548
12  1.00  0.18  0.1686548
13  2.00  0.18  0.1751190
14  5.00  0.20  0.1632993
15 10.00  0.18  0.1686548
```

Linear SVM with "best" cost C

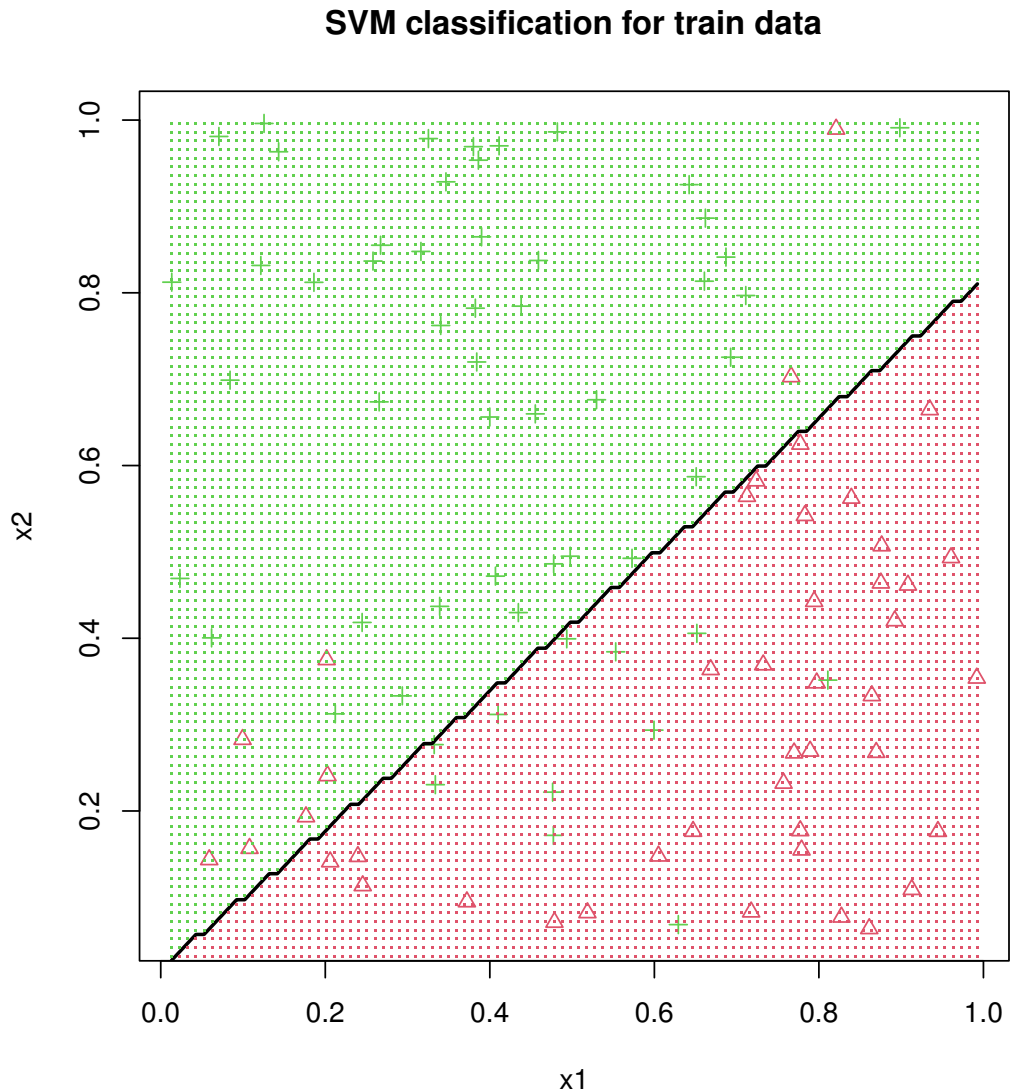
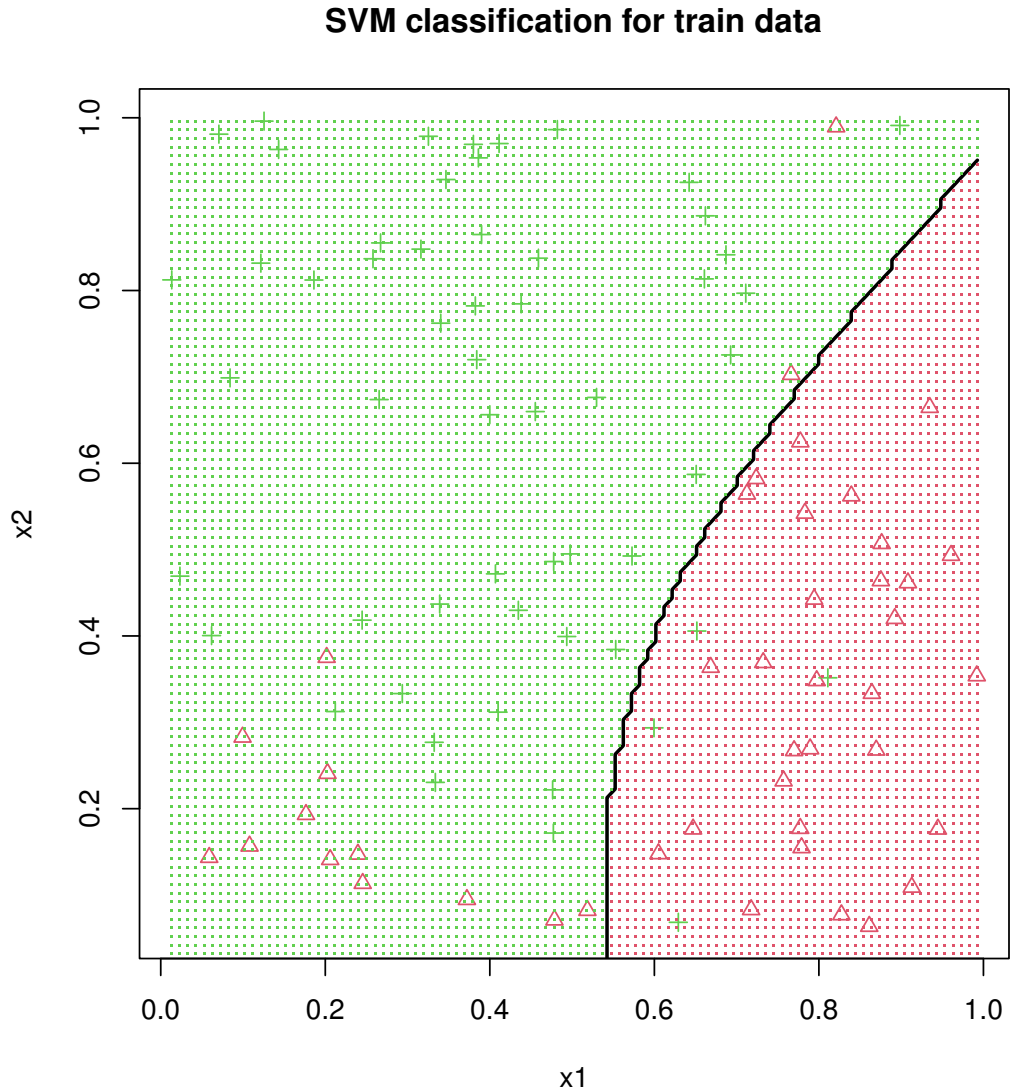


Figure: Code is available on the blackboard

Misclassification errors for testing data

```
> ## Misclassification errors for testing data
> err = vector()
> for(c in C){
+   sim.svm = svm(y ~ ., data = df.tr, kernel = "linear", scale = F, cost = c)
+   err = c(err,mean(predict(sim.svm, newdata = df.te) != df.te$y))
+ }
> C[which.min(err)]
[1] 0.2
> rbind(C,err)
    [,1]  [,2]  [,3] [,4]  [,5]  [,6] [,7]  [,8] [,9] [,10] [,11] [,12] [,13]
C   0.010 0.050 0.100 0.20 0.300 0.400  0.5 0.600  0.7 0.800 0.900 1.000   2.0
err 0.435 0.435 0.295 0.19 0.195 0.195  0.2 0.205  0.2 0.195 0.195 0.195   0.2
   [,14] [,15]
C   5.000  10.0
err 0.205   0.2
```

Polynomial SVM with "best" tuned cost



R codes for the previous plot

```
> ## Polynomial SVM with "best" tuned cost
> set.seed(1)
> sim2.tune = tune(svm, y ~ ., data = df.tr, kernel = "polynomial", scale = F,
+   ranges = list(cost = C))
> sim2.tune$best.parameters
  cost
14    5
> sim2.best = svm(y ~ ., data = df.tr, kernel = "polynomial", scale = F,
+   cost = as.numeric(sim2.tune$best.parameters))
> class.plot(sim2.best,data =  df,train.index = tr.ind,method = "svm",train = T)
```

Misclassification errors for testing data using polynomial SVM

```
> ## Misclassification errors for testing data using polynomial SVM
> err2 = vector()
> for(c in C){
+   sim2.svm = svm(y ~ ., data = df.tr, kernel = "polynomial", scale = F, cost = c)
+   err2 = c(err2,mean(predict(sim2.svm, newdata = df.te) != df.te$y))
+ }
> c("best cost for test data =",C[which.min(err2)])
[1] "best cost for test data =" "1"
> rbind(C,err2)
     [,1]  [,2]  [,3]  [,4]  [,5]  [,6] [,7]  [,8]  [,9] [,10] [,11] [,12]
C    0.010 0.050 0.100 0.200 0.300 0.400 0.50 0.600 0.700 0.800  0.90 1.000
err2 0.435 0.435 0.435 0.435 0.435 0.435 0.41 0.365 0.345 0.285  0.26 0.235
    [,13] [,14]  [,15]
C    2.000 5.000 10.000
err2 0.235 0.235  0.235
> #Compare to linear SVM
> rbind(C,err)
    [,1]  [,2]  [,3] [,4]  [,5]  [,6] [,7]  [,8] [,9] [,10] [,11] [,12] [,13]
C   0.010 0.050 0.100 0.20 0.300 0.400  0.5 0.600  0.7 0.800 0.900 1.000   2.0
err 0.435 0.435 0.295 0.19 0.195 0.195  0.2 0.205  0.2 0.195 0.195 0.195   0.2
   [,14] [,15]
C   5.000  10.0
err 0.205   0.2
```

SVM using Radial options

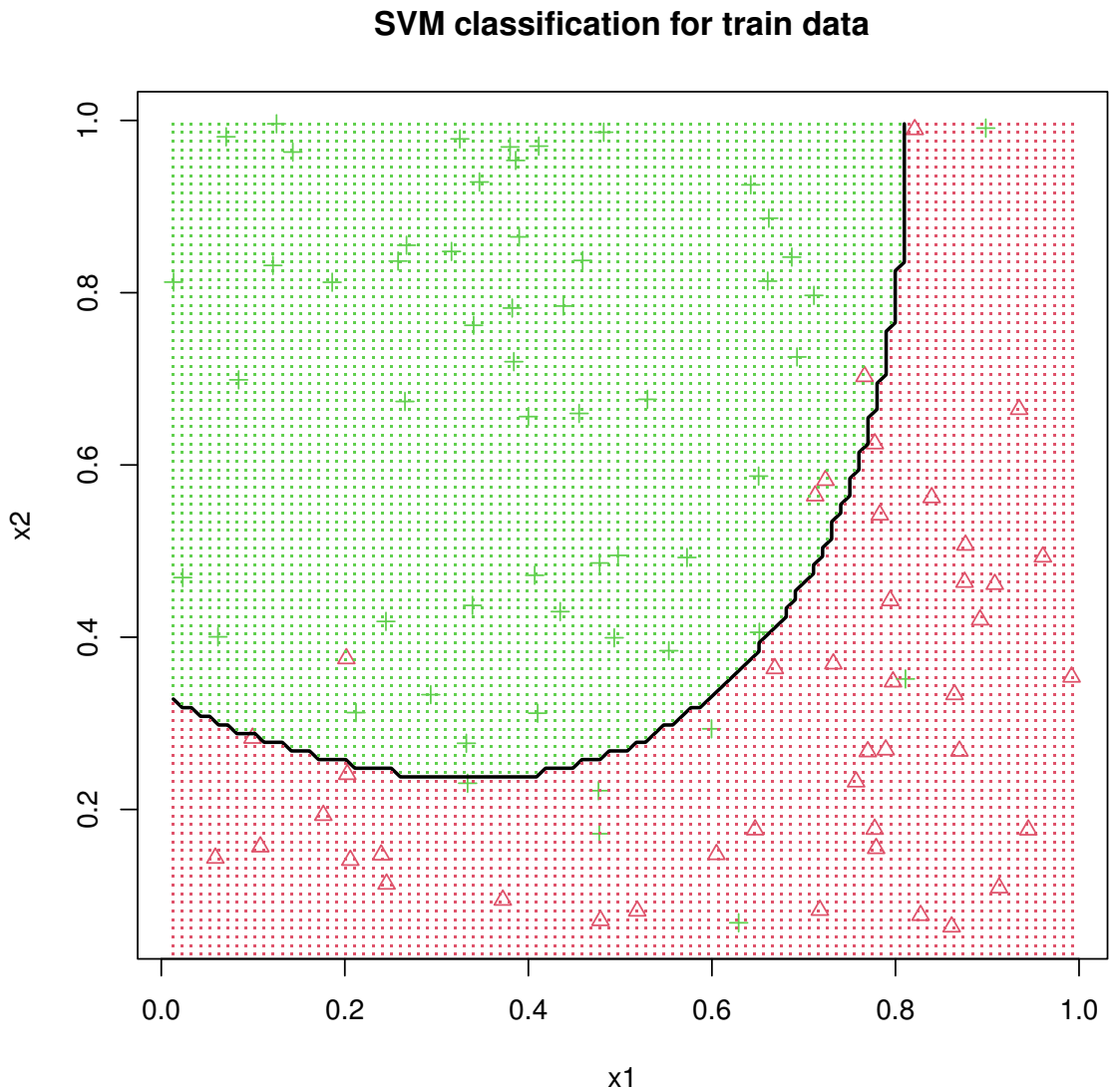
```
> ## SVM using Radial options
> set.seed(1)
> sim3.tune = tune(svm, y ~ ., data = df.tr, kernel = "radial", scale = F,
+   ranges = list(cost = C))
> sim3.tune$best.parameters
  cost
15   10
```

- It seems that we may need to explore more values of C

Exploring more values of C

```
> #It seems that we may need to explore more values of $C$
> C1 = unique(c(C,seq(6,20,2)))
> C = seq(6,20,2)
> sim3.tune = tune(svm, y ~ ., data = df.tr, kernel = "radial", scale = F,
+   ranges = list(cost = C))
> summary(sim3.tune)
Parameter tuning of 'svm':
- sampling method: 10-fold cross validation
- best parameters:
 cost
   16
- best performance: 0.13

- Detailed performance results:
  cost error dispersion
1    6  0.15 0.07071068
2    8  0.15 0.07071068
3   10  0.15 0.07071068
4   12  0.14 0.06992059
5   14  0.14 0.06992059
6   16  0.13 0.08232726
7   18  0.14 0.08432740
8   20  0.15 0.07071068
```

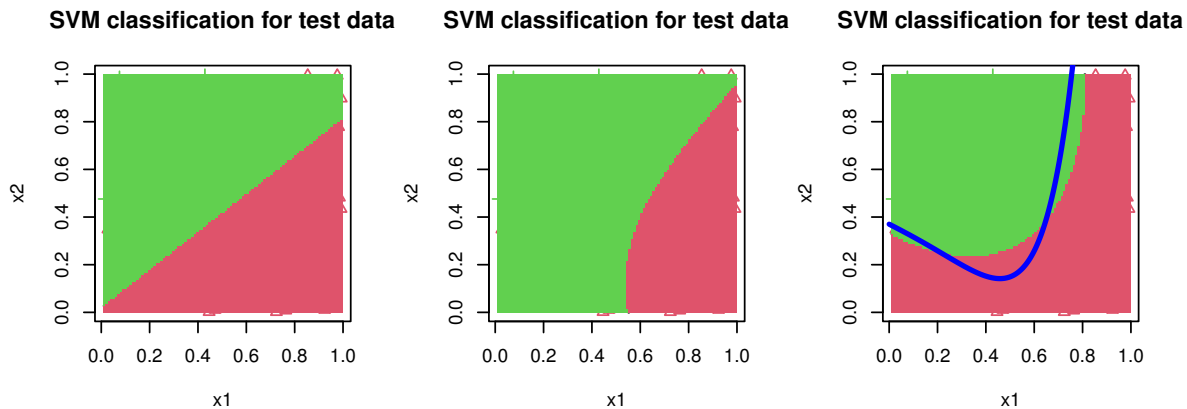
SVM with radial options and best cost



Misclassification errors for testing data with radial

```
> ## SVM with radial options and best cost
> sim3.best = svm(y ~ ., data = df.tr, kernel = "radial", scale = F, cost =
as.numeric(sim3.tune$best.parameters))
> class.plot(sim3.best,data =  df,train.index = tr.ind,method = "svm",train = T)
>
> ## Misclassification errors for testing data with radial
> C = sort(C1)
> err3 = vector()
> for(c in C){
+   sim3.svm = svm(y ~ ., data = df.tr, kernel = "polynomial", scale = F, cost = c)
+   err3 = c(err3,mean(predict(sim3.svm, newdata = df.te) != df.te$y))
+ }
> c("best cost for test data =",C[which.min(err3)])
[1] "best cost for test data =" "14"
> rbind(C,err3)
     [,1]  [,2]  [,3]  [,4]  [,5]  [,6] [,7]  [,8]  [,9] [,10] [,11] [,12]
C    0.010 0.050 0.100 0.200 0.300 0.400 0.50 0.600 0.700 0.800  0.90 1.000
err3 0.435 0.435 0.435 0.435 0.435 0.435 0.41 0.365 0.345 0.285  0.26 0.235
    [,13] [,14] [,15] [,16]  [,17]  [,18] [,19] [,20] [,21] [,22]
C    2.000 5.000 6.000 8.000 10.000 12.000 14.00 16.00 18.00 20.00
err3 0.235 0.235 0.225 0.235  0.235  0.225  0.22  0.23  0.23  0.23
```

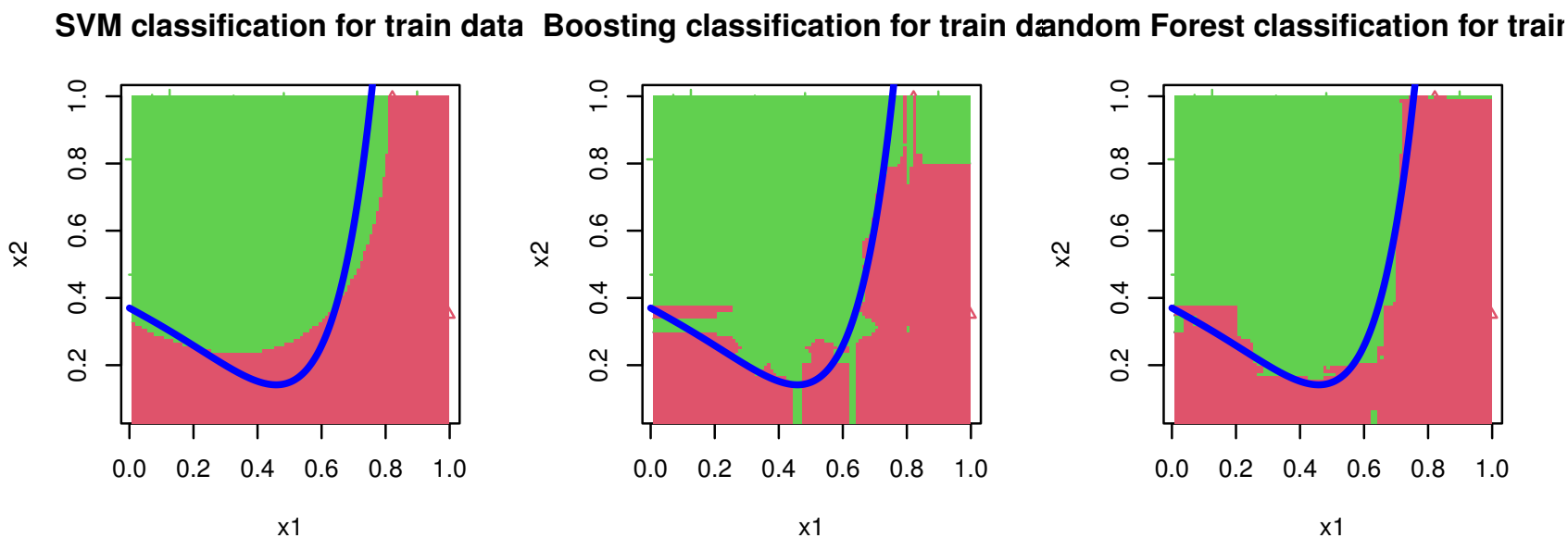
Plots for testing data



Plots for testing data

- The left, middle, and right figures are for linear, polynomial, and radial options in SVM
- The blue line in the right figure is actually the correct curve
- The data actually has a 5% random noise (with 5% chance that the correct line is incorrect)

Comparing SVM, Random Forest, and Boosting in training data



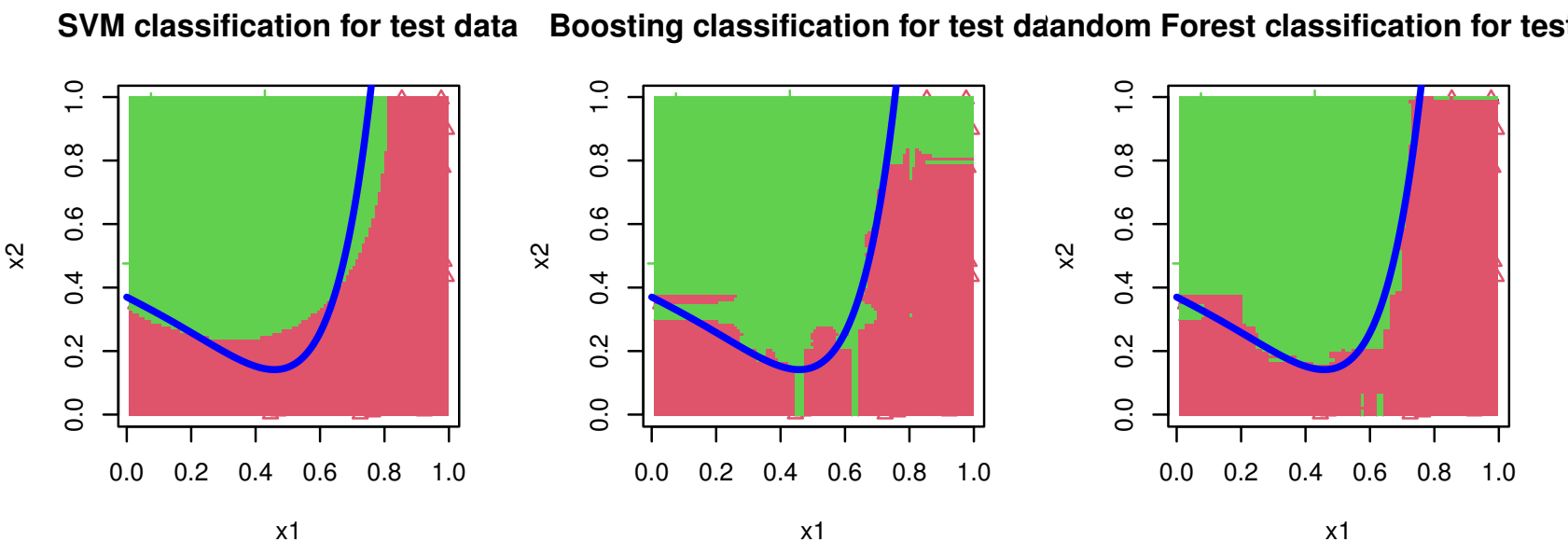
R codes for the previous plot

```
## Comparing SVM, Random Forest, and Boosting in training data
par(mfrow = c(1,3))
x = seq(0,1,length.out = 200)
par(pty = "s")
class.plot(sim3.best,data =  df,train.index = tr.ind,method = "svm",train = T)
lines(x,y.fun(x,alpha), col = "blue", lwd = 3)
sim.boost = gbm(as.numeric(y) - 1 ~ ., data = df.tr, n.trees = 5000, distribution =
"bernoulli",
  shrinkage = 0.01, interaction.depth = 4)
class.plot(sim.boost, data = df, train.index = tr.ind, train = T,method = "gbm",
  n.trees = 5000, predict_type = "response")
lines(x,y.fun(x,alpha), col = "blue", lwd = 3)
sim.rf = randomForest(y ~ ., data = df, ntree = 1000, subset = tr.ind)
class.plot(sim.rf,data = df,train.index = tr.ind,method = "randomForest", train = T)
lines(x,y.fun(x,alpha), col = "blue", lwd = 3)
```

Comparing SVM, Random Forest, and Boosting in training data

```
## Comparing SVM, Random Forest, and Boosting in training data
> c("training data error for SVM = ",mean(predict(sim3.best,df.tr) !=
as.character(df.tr$y)))
[1] "training data error for SVM = " "0.11"
> boost.pred.tr = predict(sim.boost,df.tr,n.trees = 3000, type = "response")
> c("training data error for boosting = ",mean(ifelse(boost.pred.tr>0.5,1,0) !=
as.numeric(as.character(df.tr$y))))
[1] "training data error for boosting = " "0"
> c("training data error for random forest = ",mean(predict(sim.rf,df.tr) !=
as.character(df.tr$y)))
[1] "training data error for random forest = "
[2] "0"
```

Comparing SVM, Random Forest, and Boosting in testing data



R codes for the previous plot

```
## Comparing SVM, Random Forest, and Boosting in testing data
par(mfrow = c(1,3))
x = seq(0,1,length.out = 200)
par(pty = "s")
class.plot(sim3.best,data =  df,train.index = tr.ind,method = "svm",train = F)
lines(x,y.fun(x,alpha), col = "blue", lwd = 3)
sim.boost = gbm(as.numeric(y) - 1 ~ ., data = df.tr, n.trees = 5000, distribution =
"bernoulli",
  shrinkage = 0.01, interaction.depth = 4)
class.plot(sim.boost, data = df, train.index = tr.ind, train = F,method = "gbm",
  n.trees = 5000, predict_type = "response")
lines(x,y.fun(x,alpha), col = "blue", lwd = 3)
sim.rf = randomForest(y ~ ., data = df, ntree = 1000, subset = tr.ind)
class.plot(sim.rf,data = df,train.index = tr.ind,method = "randomForest", train = F)
lines(x,y.fun(x,alpha), col = "blue", lwd = 3)
```

Comparing SVM, Random Forest, and Boosting in testing data

```
## Comparing SVM, Random Forest, and Boosting in testing data
> c("testing data error for SVM = ",mean(predict(sim3.best,df.te) !=
as.character(df.te$y)))
[1] "testing data error for SVM = " "0.1"
> boost.pred.te = predict(sim.boost,df.te,n.trees = 3000, type = "response")
> c("testing data error for boosting = ",mean(ifelse(boost.pred.te>0.5,1,0) !=
as.numeric(as.character(df.te$y))))
[1] "testing data error for boosting = " "0.145"
> c("testing data error for random forest = ",mean(predict(sim.rf,df.te) !=
as.character(df.te$y)))
[1] "testing data error for random forest = "
[2] "0.11"
```

Outline

Support vector classifiers and their extensions

- Previous discussions of the SVM concentrate on linear boundaries in the input feature space

Support vector classifiers and their extensions

- Previous discussions of the SVM concentrate on linear boundaries in the input feature space
- As with other linear methods, we can make the procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines

Support vector classifiers and their extensions

- Previous discussions of the SVM concentrate on linear boundaries in the input feature space
- As with other linear methods, we can make the procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines
- In general, linear boundaries in the enlarged space achieve better training-class separation, and translate to nonlinear boundaries in the original space

Support vector classifiers and their extensions

- Previous discussions of the SVM concentrate on linear boundaries in the input feature space
- As with other linear methods, we can make the procedure more flexible by enlarging the feature space using basis expansions such as polynomials or splines
- In general, linear boundaries in the enlarged space achieve better training-class separation, and translate to nonlinear boundaries in the original space
- Once the basis functions $h_m(\boldsymbol{x}), m = 1, \ldots, M$ are selected, the procedure is the same as before
  - We fit the SV classifier using input features $\boldsymbol{h}(\boldsymbol{x}_i) = (h_1(\boldsymbol{x}_i), \ldots, h_M(\boldsymbol{x}_i))$, for $i = 1, \ldots, n$, and produce the (nonlinear) function $\hat{f}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{x})^T \hat{\boldsymbol{\beta}} + \hat{\beta}_0$
  - The classifier is $\hat{G}(\boldsymbol{x}) = \text{sign}(\hat{f}(\boldsymbol{x}))$

Support vector classifiers and kernels

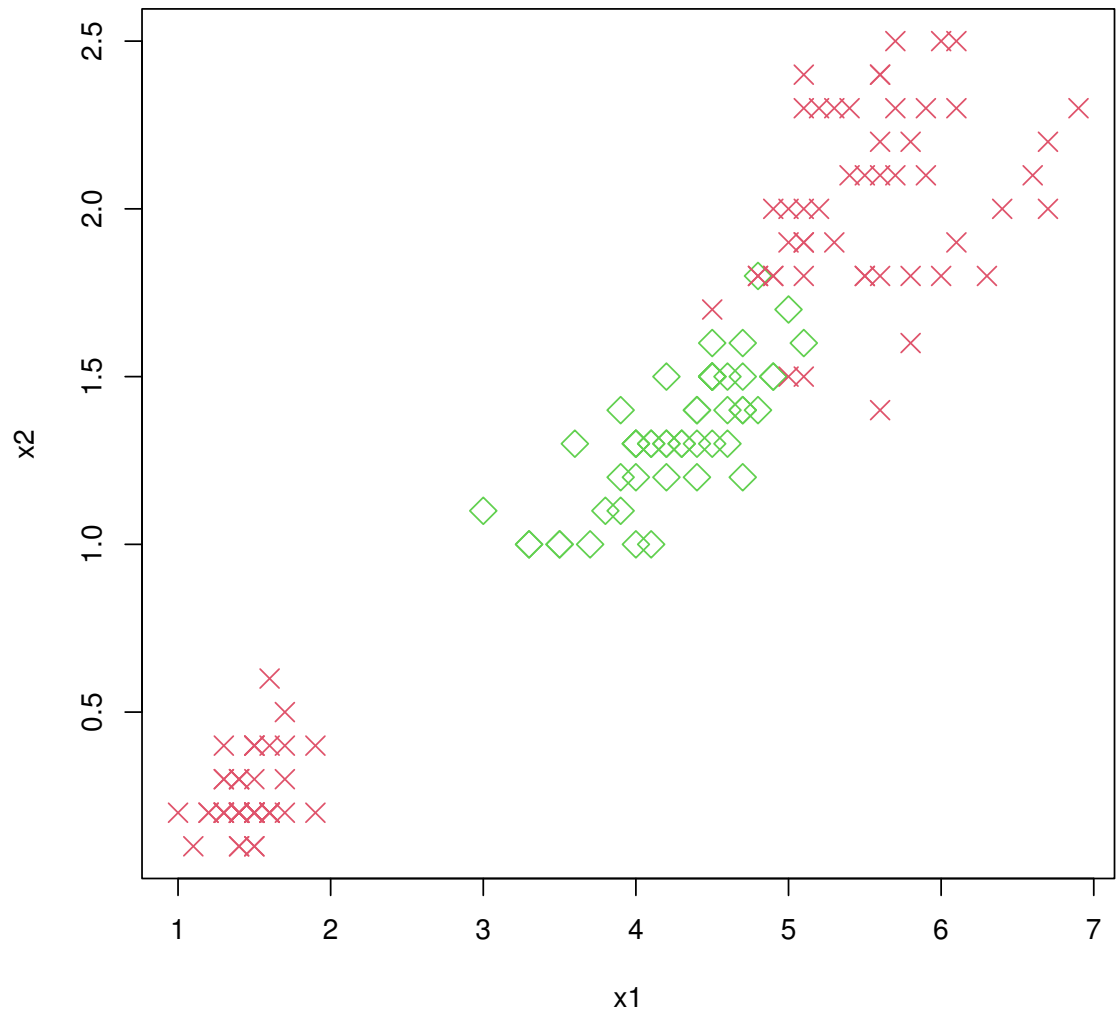
- The **support vector** machine classifier is an extension of the idea above, where the dimension of the enlarged space is allowed to get very large
  - infinite in some cases

Support vector classifiers and kernels

- The **support vector** machine classifier is an extension of the idea above, where the dimension of the enlarged space is allowed to get very large
  - infinite in some cases
- Some legitimate concerns
  - Computations would become prohibitive
  - With sufficient basis functions, the data would be separable, and overfitting would occur
- Let's see how the SVM technology deals with these issues

Support vector classifiers and kernels

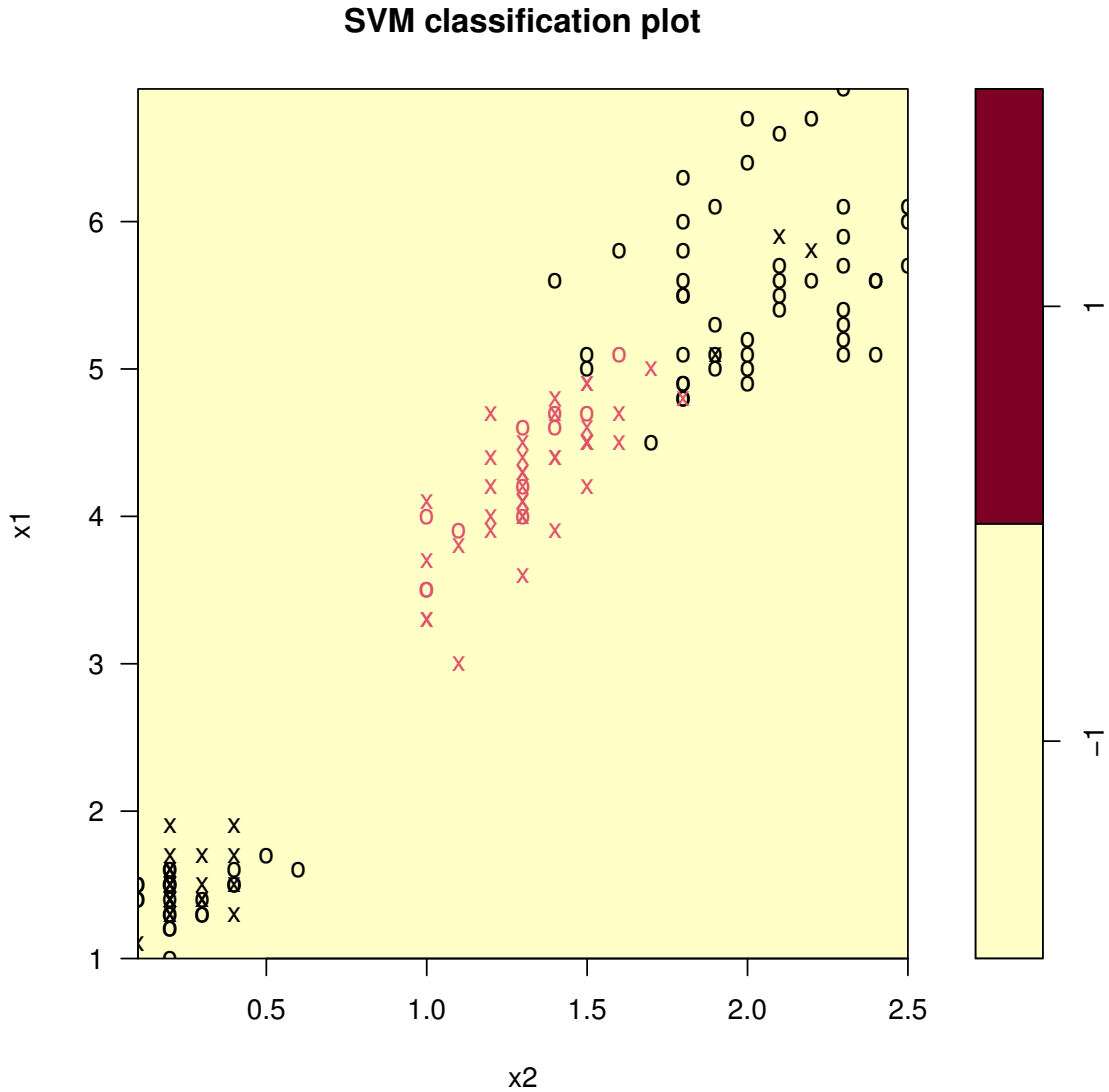
- We want to separate “Versicolor” from Non-“Versicolor” using “Petal.Length” and “Petal.Width”



R codes for the previous plot

```
# Extensions
# Iris Example, with a modification
iris2 = iris[,3:5]
colnames(iris2) = c("x1","x2","y")
iris2$y = as.factor(ifelse(iris2$y == "versicolor",1,-1))
iris2$z = sqrt((iris2$x1-mean(iris2$x1))^2 + (iris2$x2 - mean(iris2$x2))^2)
set.seed(1)
tr.ind = sample(seq_len(nrow(iris2)),0.7*nrow(iris2))
iris2.tr = iris2[tr.ind,]
iris2.te = iris2[-tr.ind,]
plot(iris2[,1:2],col = as.numeric(iris2$y)+1, pch = as.numeric(iris2$y)+3, cex = 1.5)
```

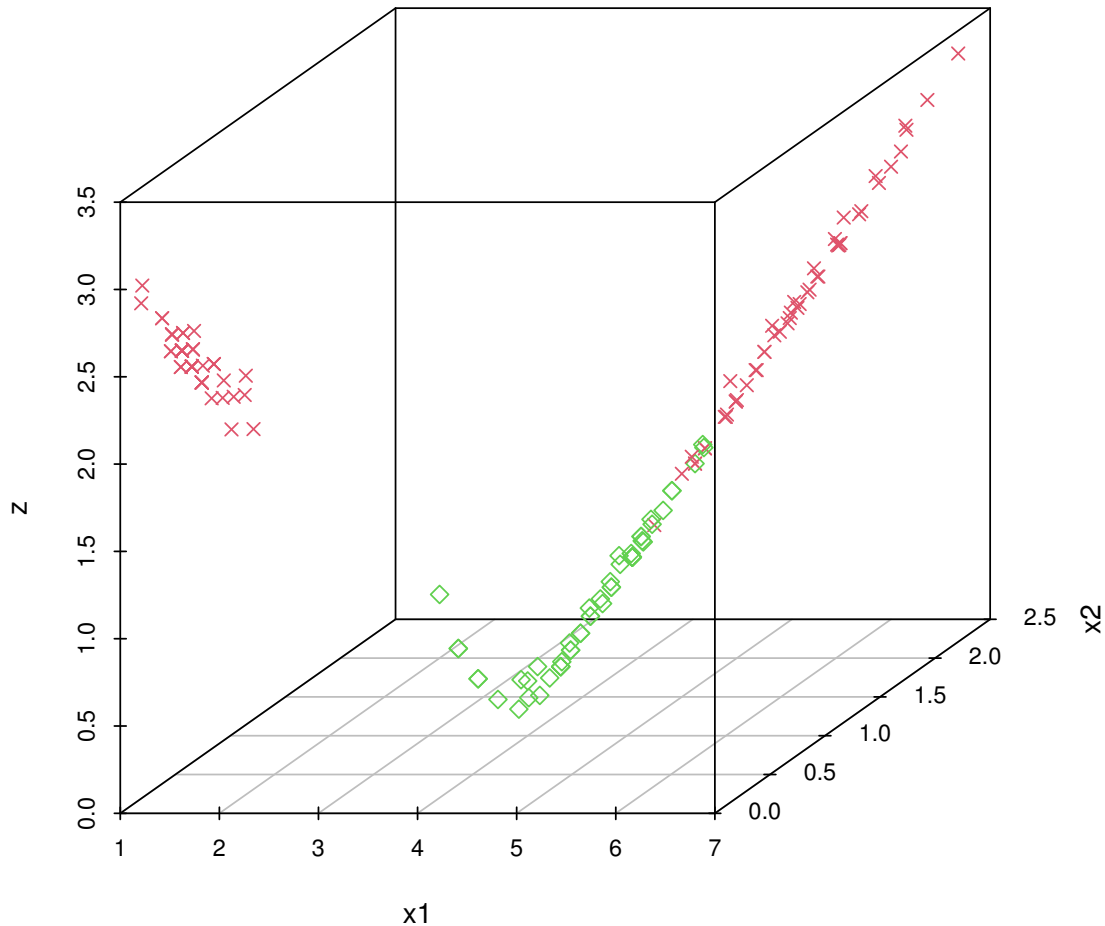
Use a direct linear SVM



R codes for the previous plot

```
> ## Use a direct linear SVM
> set.seed(1)
> C = c(seq(0.1,1,0.1),seq(2,10,1),seq(20,100,10))
> iris2.tune = tune(svm, y ~ x1 + x2, data = iris2.tr, kernel = "linear",ranges = list(cost =
C))
> c("best tuned cost variable = ",as.numeric(iris2.tune$best.parameters))
[1] "best tuned cost variable = " "0.1"
> iris2.best = svm(y ~ x1 + x2, data = iris2.tr, kernel = "linear", scale = F, cost =
as.numeric(iris2.tune$best.parameters))
> iris2.tr.lin = predict(iris2.best, iris2.tr)
> iris2.te.lin = predict(iris2.best,iris2.te)
> lin.tr.err = mean(iris2.tr.lin != iris2.tr$y)
> lin.te.err = mean(iris2.te.lin != iris2.te$y)
> c("training data error = ",lin.tr.err)
[1] "training data error = " "0.314285714285714"
> c("testing data error = ",lin.te.err)
[1] "testing data error = " "0.377777777777778"
> plot(iris2.best,iris2,x1 ~ x2)
>
```

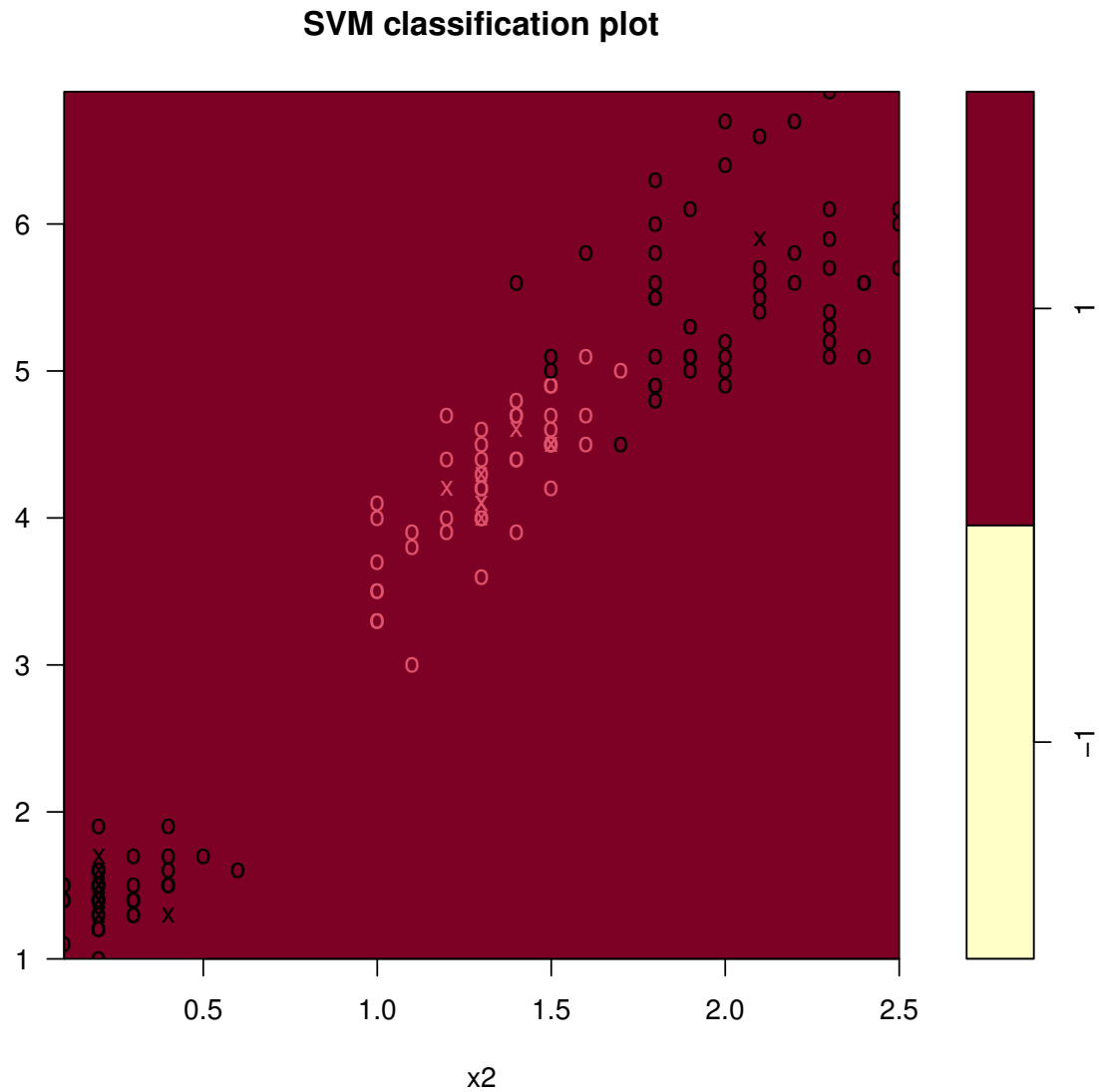
Adding a new feature variable



R codes for the previous plot

```
> ## Adding a new feature variable
> ##Let $z=\sqrt{(x_1-\bar{x}_1)^2 + (x_2-\bar{x}_2)^2}$
>
> with(data = iris2,scatterplot3d(x = x1,y = x2,z = z,color = as.numeric(y)+1,
+   pch = as.numeric(y)+3))
>
```

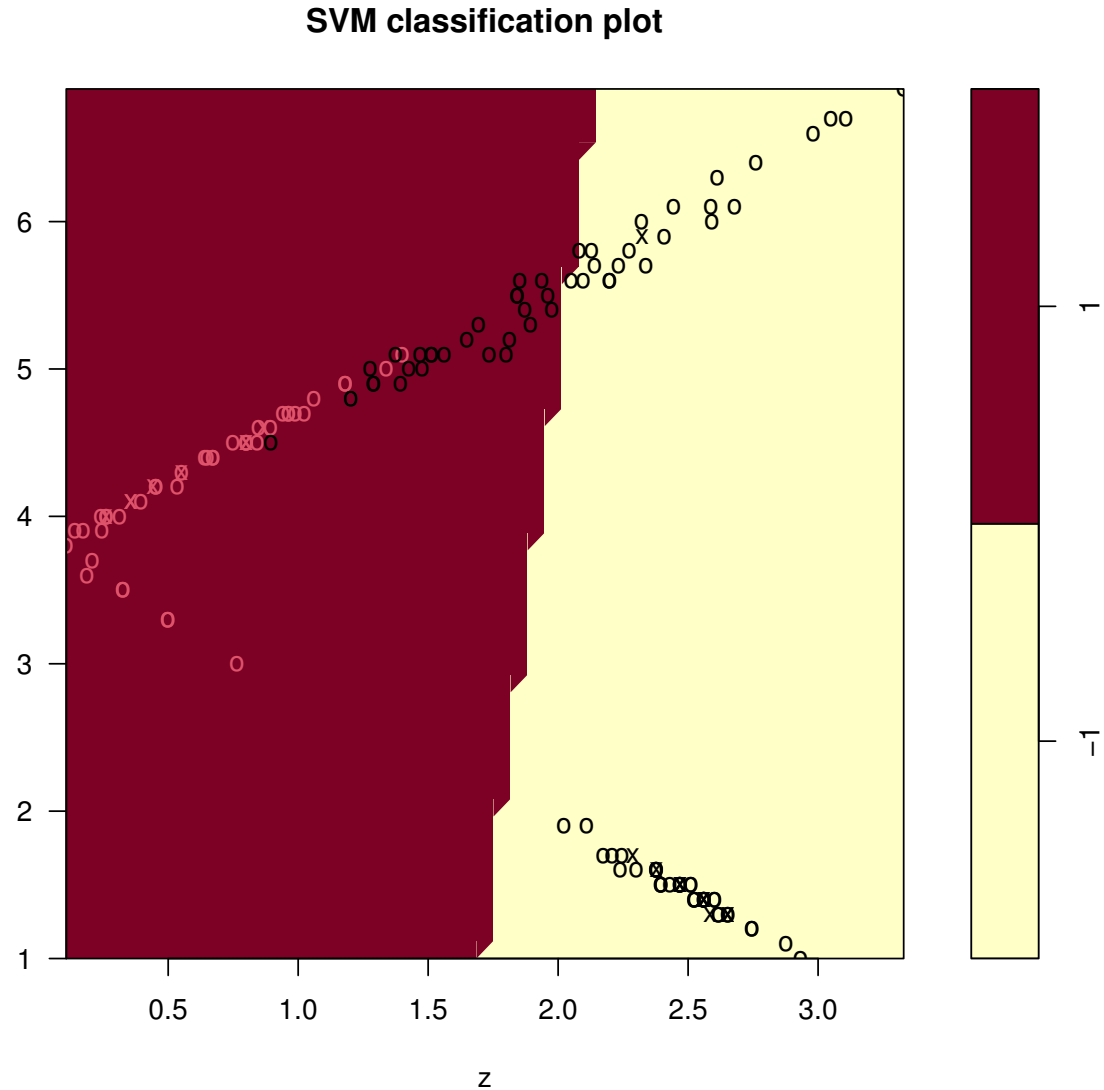
Redo a linear SVM with the added feature



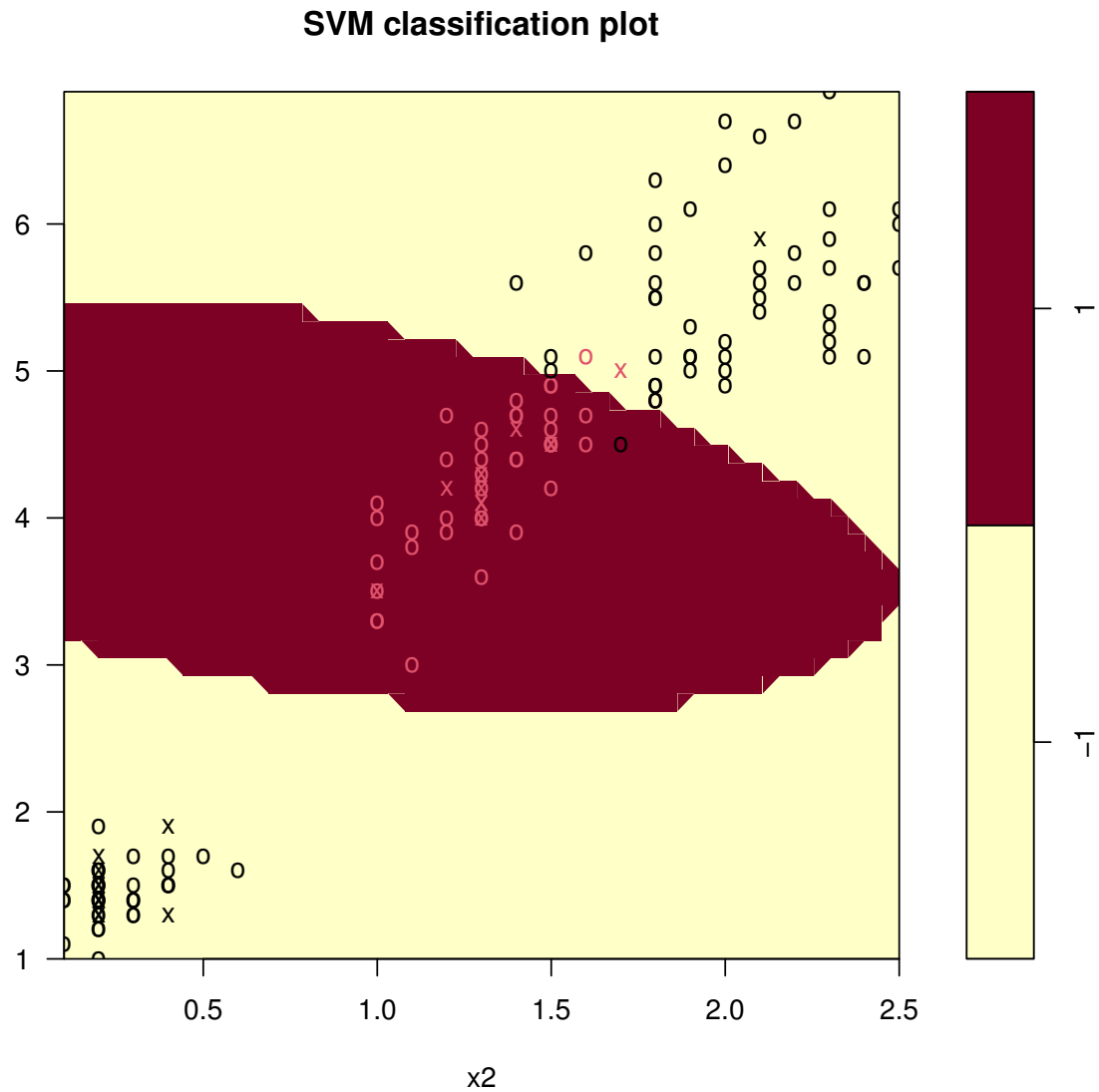
R codes for the previous plot

```
## Redo a linear SVM with the added feature
set.seed(1)
iris2.tune2 = tune(svm, y ~ ., data = iris2.tr, kernel = "linear",ranges = list(cost = C))
c("best tuned cost variable = ",as.numeric(iris2.tune2$best.parameters))
iris2.best2 = svm(y ~ ., data = iris2.tr, kernel = "linear", scale = F, cost =
as.numeric(iris2.tune2$best.parameters))
iris2.tr.lin2 = predict(iris2.best2, iris2.tr)
iris2.te.lin2 = predict(iris2.best2,iris2.te)
lin.tr.err2 = mean(iris2.tr.lin2 != iris2.tr$y)
lin.te.err2 = mean(iris2.te.lin2 != iris2.te$y)
c("training data error = ",lin.tr.err2)
c("testing data error = ",lin.te.err2)
plot(iris2.best2,iris2,x1 ~ x2)
```

SVM plot from a different angle



SVM plot from a different angle



R codes for the previous plots

```
> ## SVM plot from a different angle
> plot(iris2.best2,iris2,x1 ~ z)
>
> ## SVM with a different option: Radial
> iris2.svm = svm(y ~ ., data = iris2.tr[,1:3], kernel = "radial", scale = F)
> iris2.tr.rad = predict(iris2.svm, iris2.tr[,1:3])
> iris2.te.rad = predict(iris2.svm, iris2.te[,1:3])
> rad.tr.err = mean(iris2.tr.rad != iris2.tr$y)
> rad.te.err = mean(iris2.te.rad != iris2.te$y)
> c("training data error = ",rad.tr.err)
[1] "training data error = " "0.0380952380952381"
> c("testing data error = ",rad.te.err)
[1] "testing data error = " "0.0222222222222222"
> plot(iris2.svm,iris2[,1:3])
```

Classification with Non-linear Decision Boundaries

- Enlarge the space of features by including transformations; e.g. $X_1^2, X_1^3, X_1 X_2, X_1 X_2^2, \ldots$.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

Classification with Non-linear Decision Boundaries

- Enlarge the space of features by including transformations; e.g. $X_1^2, X_1^3, X_1X_2, X_1X_2^2, \ldots$.
- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the original space.

For example, suppose we use $h(X) = (X_1, X_2, X_1^2, X_2^2, X_1X_2)^T$ instead of just $X = (X_1, X_2)^T$. Then the decision boundary would be of the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 = \beta_0 + \beta^T h(X) = 0,$$

which leads to non-linear decision boundaries in the original space.

Support Vector Machine (SVM)

- SVM is an extension of the support vector classifier that results from enlarging the feature space via *kernels*.
- The kernel approach is a more elegant and controlled way to introduce nonlinearities in support vector classifiers.
- It turns out that the solution to the support vector classifier problem involves only the *inner products* of the observations.
- The inner product of two r-vectors a and b is defined as $\langle a, b\rangle = \sum_{i=1}^{r} a_i b_i$, which is just $a^T b$.

Solution to the Support Vector Classifier Problem

- This optimization problem is equivalent to

$$\min_{\beta_0,\beta,\epsilon_1,\ldots,\epsilon_n} \frac{1}{2}\|\beta\|^2 + C^* \sum_{i=1}^{n} \epsilon_i \text{ subject to}$$

$$\epsilon_i \geq 0,\ y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq 1 - \epsilon_i \quad \forall i.$$

where the parameter $C^* = \infty$ corresponds to $C = 0$.

Solution to the Support Vector Classifier Problem

- This optimization problem is equivalent to

$$\min_{\beta_0,\beta,\epsilon_1,\ldots,\epsilon_n} \frac{1}{2}\|\beta\|^2 + C^* \sum_{i=1}^{n} \epsilon_i \text{ subject to}$$

$$\epsilon_i \geq 0,\ y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq 1 - \epsilon_i \quad \forall i.$$

  where the parameter $C^* = \infty$ corresponds to $C = 0$.
- This is can be solved using Lagrange multipliers

$$L_P = \frac{1}{2}\|\beta\|^2 + C^* \sum_{i=1}^{n} \epsilon_i - \sum_{i=1}^{n} \alpha_i \left[y_i(\beta_0 + \beta^T x_i) - (1 - \epsilon_i)\right] - \sum_{i=1}^{n} \mu_i \epsilon_i.$$

  which we minimize w.r.t β, β_0 and ϵ_i.
- Setting the respective derivatives to zero, we get

$$\beta = \sum_{i=1}^{n} \alpha_i y_i x_i, \quad 0 = \sum_{i=1}^{n} \alpha_i y_i, \quad \alpha_i = C^* - \mu_i, \forall i \qquad (4)$$

  as well as the positivity constraints $\alpha_i, \mu_i, \epsilon_i \geq 0$.

Solution to the Support Vector Classifier Problem

- Plugging these equations to L_P, we obtain

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} \langle x_i, x_{i'} \rangle, \tag{5}$$

  and $\hat{\alpha}_i$ is obtained by maximizing L_D under the constraints $0 \leq \alpha_i \leq C^*$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$.
- The Karush-Kuhn-Tucker conditions include constraints

$$\alpha_i \Big[y_i(\beta_0 + \beta^T x_i) - (1 - \epsilon_i) \Big] = 0, \tag{6}$$

$$\mu_i \epsilon_i = 0, \tag{7}$$

$$y_i(\beta_0 + \beta^T x_i) - (1 - \epsilon_i) \geq 0. \tag{8}$$

- The equations (4)-(8) uniquely characterize the solution.
- From (4), the support vector classifier takes the form

$$f(x) = \beta_0 + \beta^T x = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x \rangle.$$

Support Vector Machine

- To obtain non-linear boundaries, we had fit the SV classifier using transformed feature vector, say, $h(x_i)$.
- Then we can see that the solution to SV classifier involve $h(x_i)$ only through inner products $\langle h(x_i), h(x_{i'})\rangle$.

Support Vector Machine

- To obtain non-linear boundaries, we had fit the SV classifier using transformed feature vector, say, $h(x_i)$.
- Then we can see that the solution to SV classifier involve $h(x_i)$ only through inner products $\langle h(x_i), h(x_{i'})\rangle$.
- In fact, we need not specify the transformation $h(x_i)$ at all, some special *kernel functions* can do this for us. E.g.

$$K(x_i, x_{i'}) = \left(1 + \sum_{j=1}^{p} x_{ij}x_{i'j}\right)^d .$$

  This is known as a *polynomial kernel of degree d*.
- The solution has the form

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i y_i K(x, x_i).$$

Support Vector Machine

- Another popular choice for K is the *radial kernel*

$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right).$$

- This kernel has very local behavior, in the sense that only nearby training observations have an effect on the class label of a test observation.

Support Vector Machine

- Another popular choice for K is the *radial kernel*

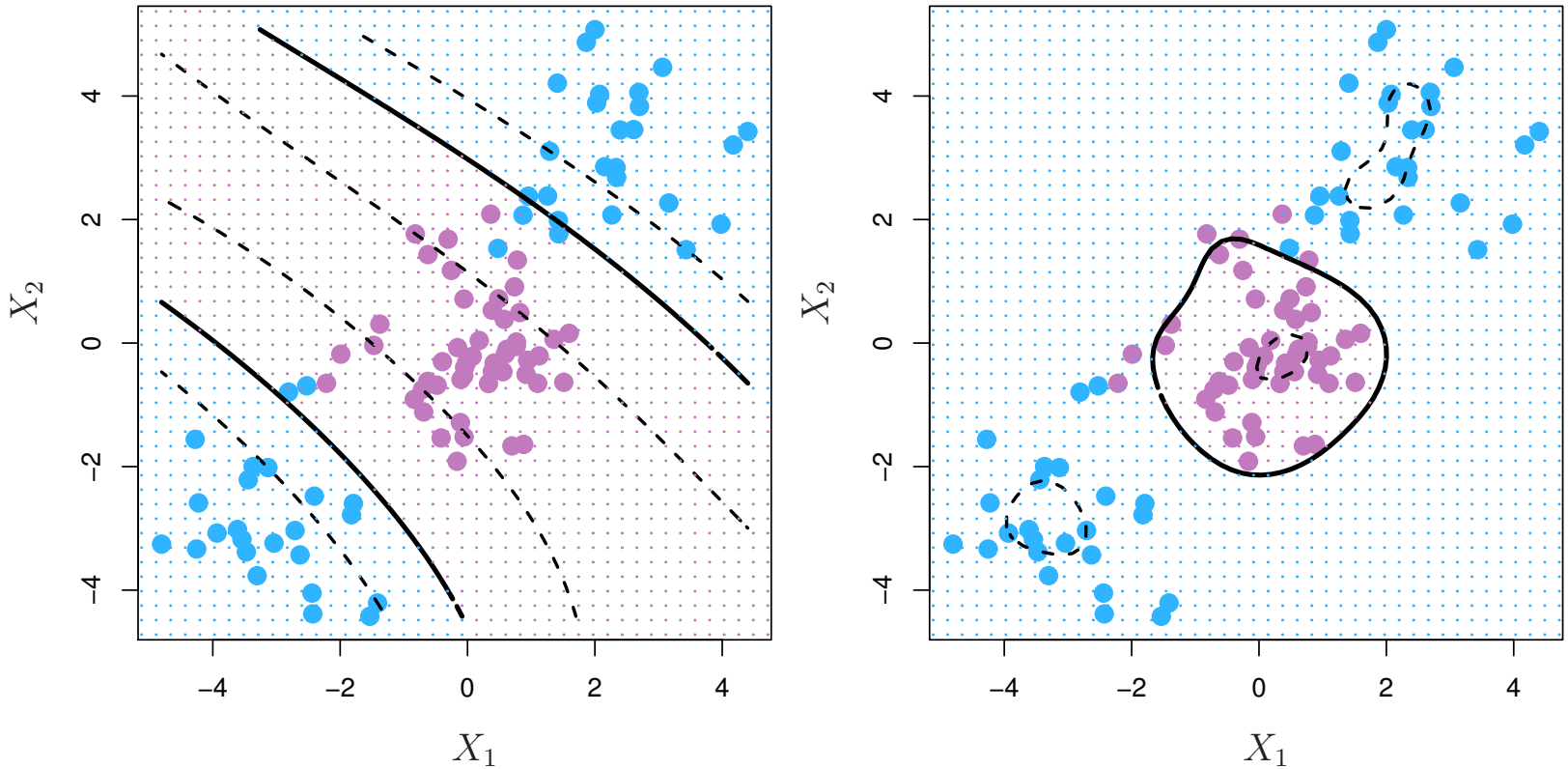
$$K(x_i, x_{i'}) = \exp\left(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\right).$$

- This kernel has very local behavior, in the sense that only nearby training observations have an effect on the class label of a test observation.
- **Fact**: It turns out that most of the α_i can be zero:

$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i y_i K(x, x_i),$$

where $\mathcal{S}$ is the support set of indices i such that $\alpha_i > 0$. Note α_i is nonzero only for the support vectors in the solution—that is, if a training observation is not a support vector, then its α_i equals zero. Why?

Support Vector Machine: An Example



Left: *An SVM with a polynomial kernel of degree 3 is applied.*
Right: *An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.*

Advantage of Using a Kernel

- One advantage is computational, and it amounts to the fact that using kernels, one need only compute $K(x_i, x_{i'})$ for all distinct pairs i, i'. This can be done without explicitly working in the enlarged feature space.
- This is important because in many applications of SVMs, the enlarged feature space is so large that computations are intractable.
- For some kernels, such as the radial kernel, the feature space is implicit and infinite-dimensional, so we could never do the computations there anyway!

Outline

SVMs with $K > 2$ Classes

- One-Versus-One (OVO). Fit all $\binom{K}{2}$ pairwise SVM classifiers. Classify test observation x^* to the class to which it was most frequently assigned in these $\binom{K}{2}$ pairwise classifications.

SVMs with $K > 2$ Classes

- One-Versus-One (OVO). Fit all $\binom{K}{2}$ pairwise SVM classifiers. Classify test observation x^* to the class to which it was most frequently assigned in these $\binom{K}{2}$ pairwise classifications.
- One-Versus-All. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.

SVMs with $K > 2$ Classes

- One-Versus-One (OVO). Fit all $\binom{K}{2}$ pairwise SVM classifiers. Classify test observation x^* to the class to which it was most frequently assigned in these $\binom{K}{2}$ pairwise classifications.
- One-Versus-All. Fit K different 2-class SVM classifiers $\hat{f}_k(x)$, $k = 1, \ldots, K$; each class versus the rest. Classify x^* to the class for which $\hat{f}_k(x^*)$ is largest.
- Which to choose? If K is not too large, use OVO.

Outline

Relationship to Logistic Regression

- The SV classifier optimization problem is equivalent to

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \left\{ \sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}, \qquad (9)$$

  where λ is a nonnegative tuning parameter. Why?

Relationship to Logistic Regression

- The SV classifier optimization problem is equivalent to

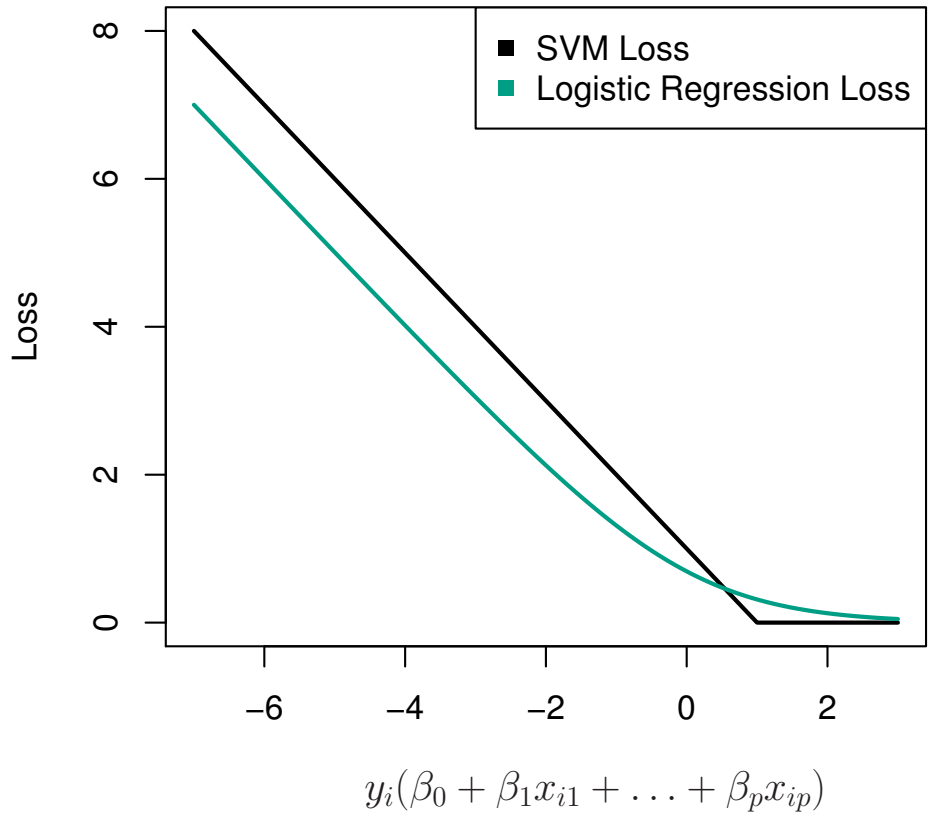
$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \left\{ \sum_{i=1}^{n} \max[0, 1 - y_i f(x_i)] + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}, \qquad (9)$$

  where λ is a nonnegative tuning parameter. Why?
- This has the "Loss+Penalty" form

$$\min_{\beta_0,\beta_1,\ldots,\beta_p} \{L(\mathbf{X}, \mathbf{y}, \beta) + \lambda P(\beta)\},$$

- In the case of (9) the loss function is known as *hinge loss*, very similar to the loss function in logistic regression.

SVM Loss vs. LR Loss



- When $y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) > 1$, then the SVM loss is zero, since this corresponds to an observation that is on the correct side of the margin.

SVM or Logistic Regression?

- When classes are (nearly) separable, SVM does better than LR. So does LDA.
- When not, LR (with ridge penalty) and SVM very similar.
- If you wish to estimate probabilities, LR is the choice.
- For nonlinear boundaries, kernel SVMs are popular. Can use kernels with LR and LDA as well, but computations are more expensive.

A simulated linear regression example

- The data set has two X variables and the response is

$$E(Y) = 2x_1 + 5x_2 - 2x_1x_2 - x_1^2 + 100\sin(x_1),$$

with standard error 0.1

A simulated linear regression example

- The data set has two X variables and the response is

$$E(Y) = 2x_1 + 5x_2 - 2x_1x_2 - x_1^2 + 100\sin(x_1),$$

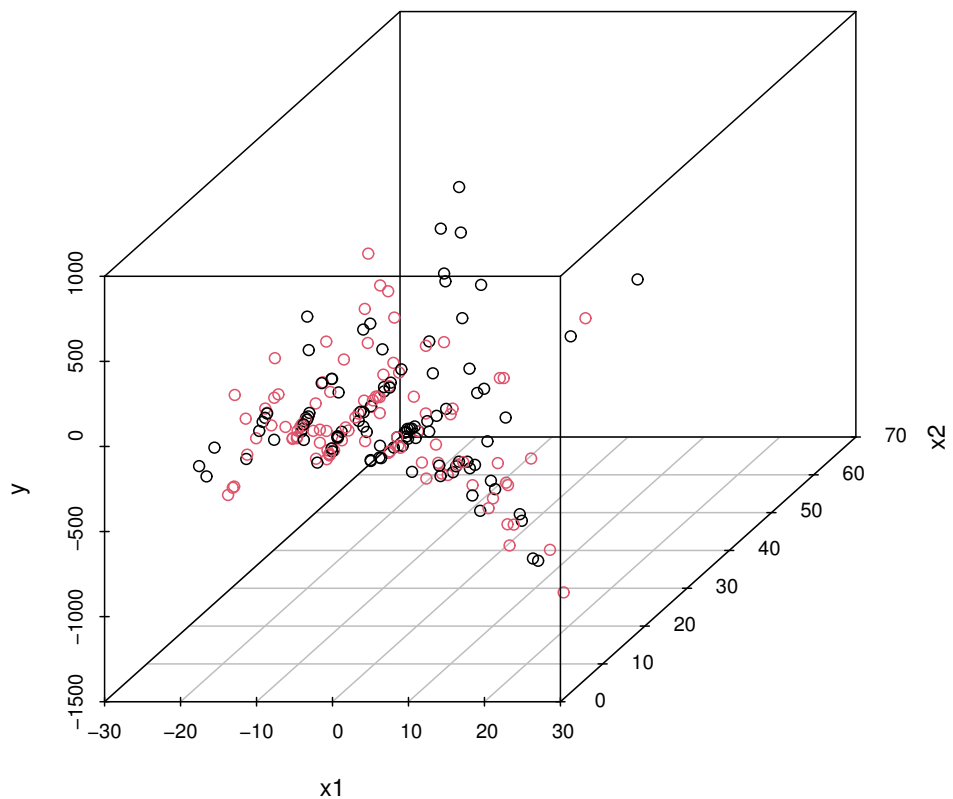
  with standard error 0.1
- 100 training data (black) and 100 test data (red)

A simulated linear regression example

- The data set has two X variables and the response is

$$E(Y) = 2x_1 + 5x_2 - 2x_1x_2 - x_1^2 + 100\sin(x_1),$$

  with standard error 0.1
- 100 training data (black) and 100 test data (red)



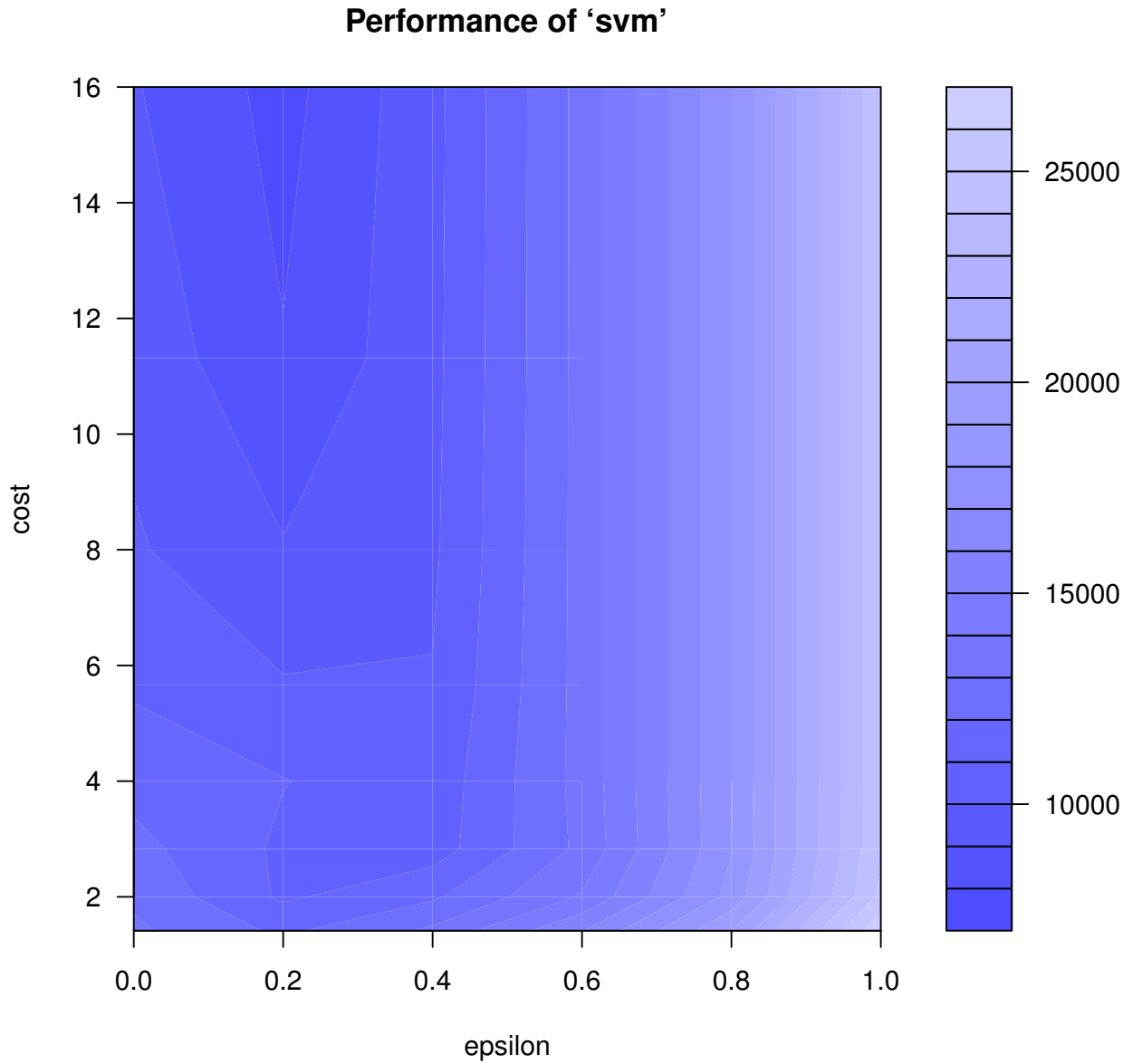
R code for the previous plot

```
set.seed(1)
x1 = rnorm(200,1,10)
x2 = rexp(200,0.1)
y = sin(x1)*100 -x1^2 + 2*x1 + 5*x2 - 2*x1*x2 + rnorm(200,0,0.1)
df = data.frame(x1 = x1,x2 = x2,y = y)
tr.ind = 1:100
df.tr = df[tr.ind,]
df.te = df[-tr.ind,]
df$m = c(rep(1,100),rep(2,100))
with(data = df, scatterplot3d(x = x1, y = x2, z = y, color = m))
df$m = NULL
```

Different methods with root MSEs

```
> ## Different methods with root MSEs
> sim.lm = lm(y ~ ., data = df.tr)
> sqrt(mean((predict(sim.lm,newdata = df.te) - df.te$y)^2))
[1] 180.1191
> sim.poly = lm(y ~ poly(x1,3)+poly(x2,3), data = df.tr)
> sqrt(mean((predict(sim.poly,newdata = df.te) - df.te$y)^2))
[1] 146.4926
> sim.gam = mgcv::gam(y ~ s(x1) + s(x2), data = df, method = "REML", subset = tr.ind)
> sqrt(mean((predict(sim.gam,newdata = df.te) - df.te$y)^2))
[1] 143.9391
> sim.svm.l = svm(y ~ ., data = df, kernel = "linear", scale = F, subset = tr.ind)
> sqrt(mean((predict(sim.svm.l,newdata = df.te) - df.te$y)^2))
[1] 193.0416
> sim.svm.r = svm(y ~ ., data = df, kernel = "radial", scale = F, subset = tr.ind)
> sqrt(mean((predict(sim.svm.r,newdata = df.te) - df.te$y)^2))
[1] 261.2313
> set.seed(12345)
> sim.svm.tune = tune(svm, y ~ ., data = df.tr,
+   ranges = list(epsilon = seq(0,1,0.2), cost = 2^(seq(0.5,4,.5))))
> #plot(sim.svm.tune)
> sqrt(mean((predict(sim.svm.tune$best.model,newdata = df.te) - df.te$y)^2))
[1] 86.08769
```

Performance of 'SVM'



Outline

R Resources

- Sections 9.6 on the textbook.

References

- *An Introduction to Statistical Learning with Applications in R*, by James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). `http://www-bcf.usc.edu/~gareth/ISL/`.
- *The Elements of Statistical Learning, 2nd Edition*, by Hastie, T., Tibshirani, R. and Friedman, J. (2009). `http://statweb.stanford.edu/~tibs/ElemStatLearn/`
- Slides and videos for Statistical Learning MOOC by Hastie, T. and Tibshirani, R. ▸ Link
- Slides and video tutorials related to this book by Abass Al Sharif. `http://www-bcf.usc.edu/~gareth/ISL/`.