Annotations made in class 19.03.2018

TMA4268 Statistical Learning V2018

Module 9: SUPPORT VECTOR MACHINES

Mette Langaas and Thea Roksvåg, Department of Mathematical Sciences, NTNU

week 12, version 18.03.2018

Before we start

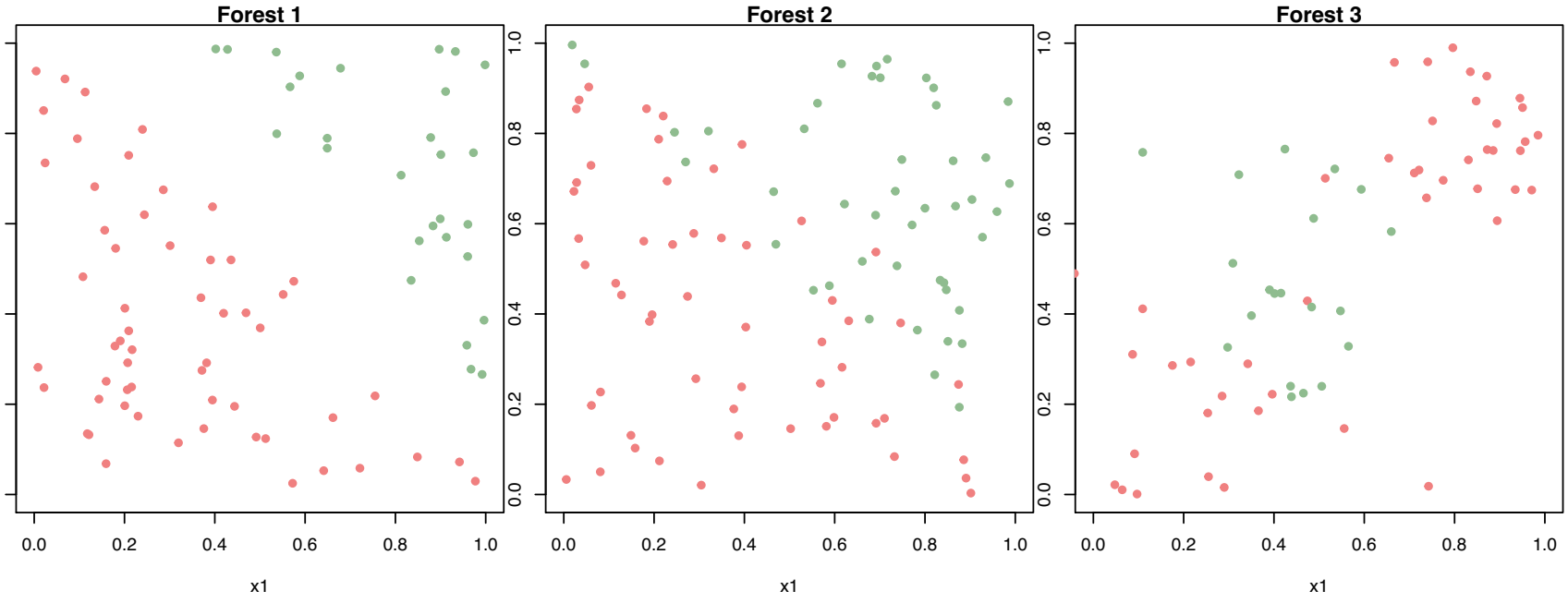
Learning material

- James et al (2013): An Introduction to Statistical Learning. Chapter 9.
- Classnotes 19.03.2018

Some of the figures in this presentation are taken from (or are inspired by) "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

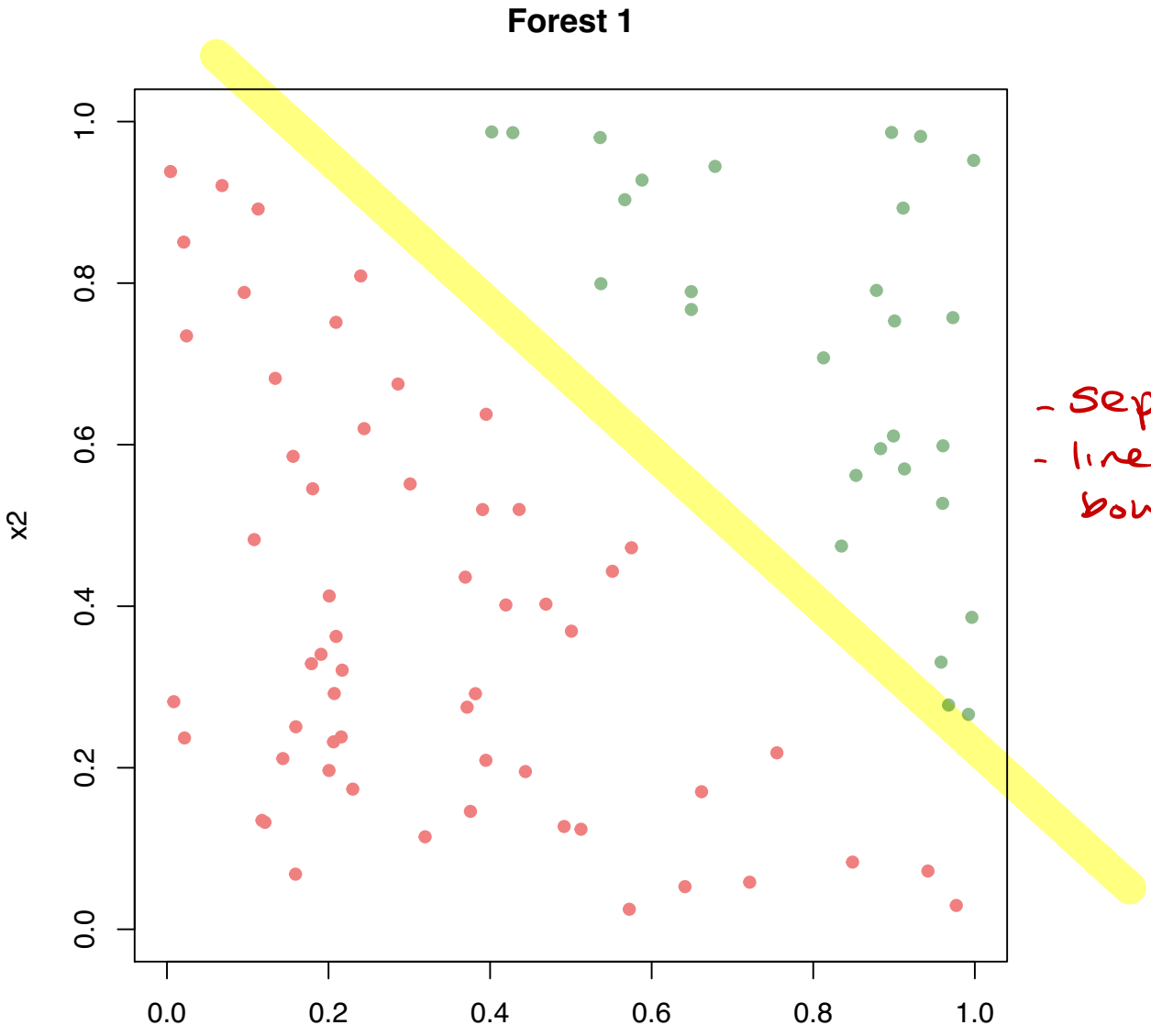
Motivation

Suppose that you are interested in the distribution of two tree types: redwood and pines. You have three different study areas in which these trees grow. Your study areas are visualized in the three figures below with orange points indicating the position of a redwood tree and green points indicating the position of a pine tree in a forest.



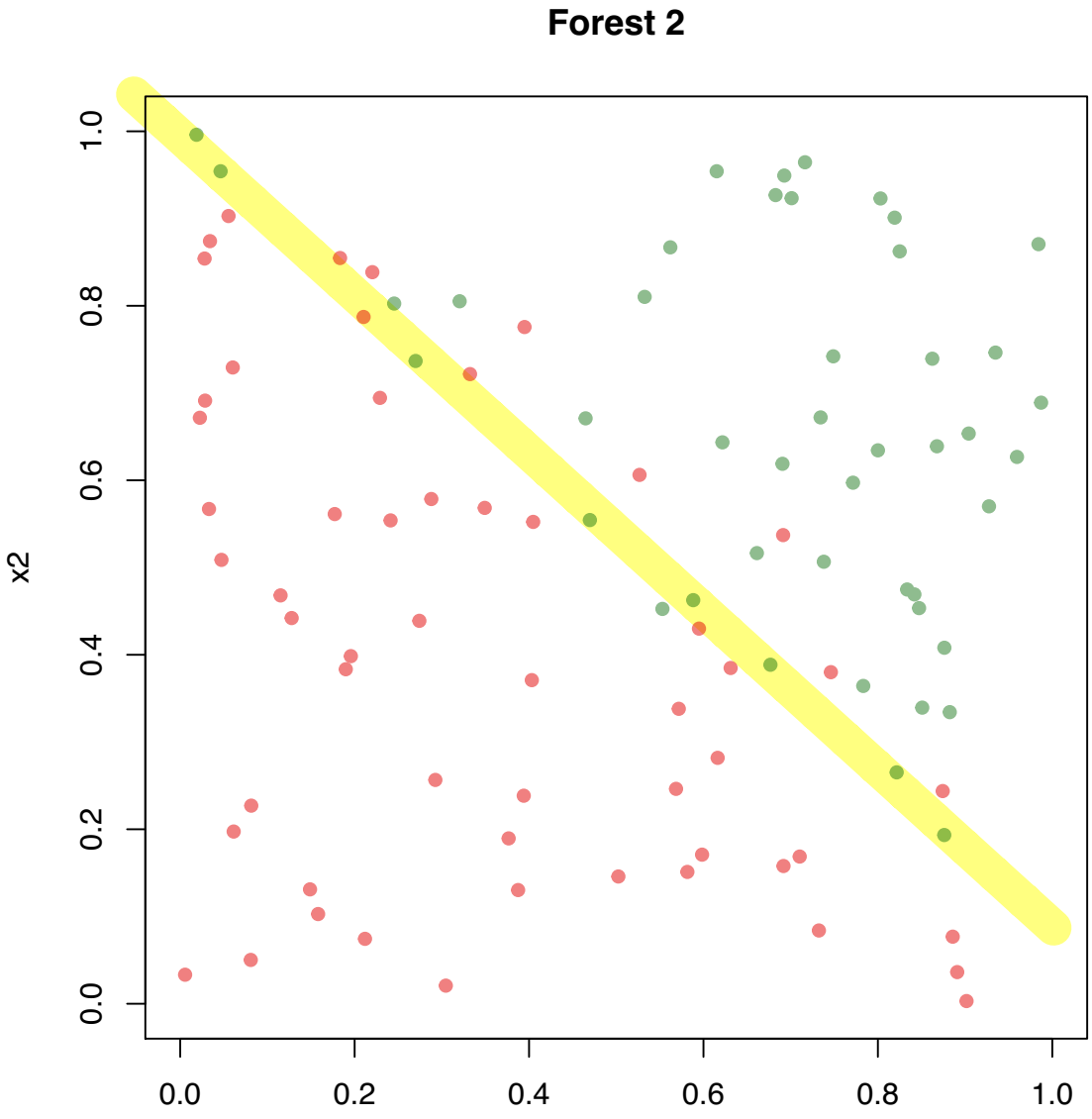
Assume you want to build one continuous fence to separate the two tree types in each of the three study areas. **Where should you build the fence?**

Forest 1 illustrates the problem of finding an **optimal separating hyperplane** for a dataset. In this module you are going to learn a method for finding optimal hyperplanes called **Maximal Margin hyperplanes**.



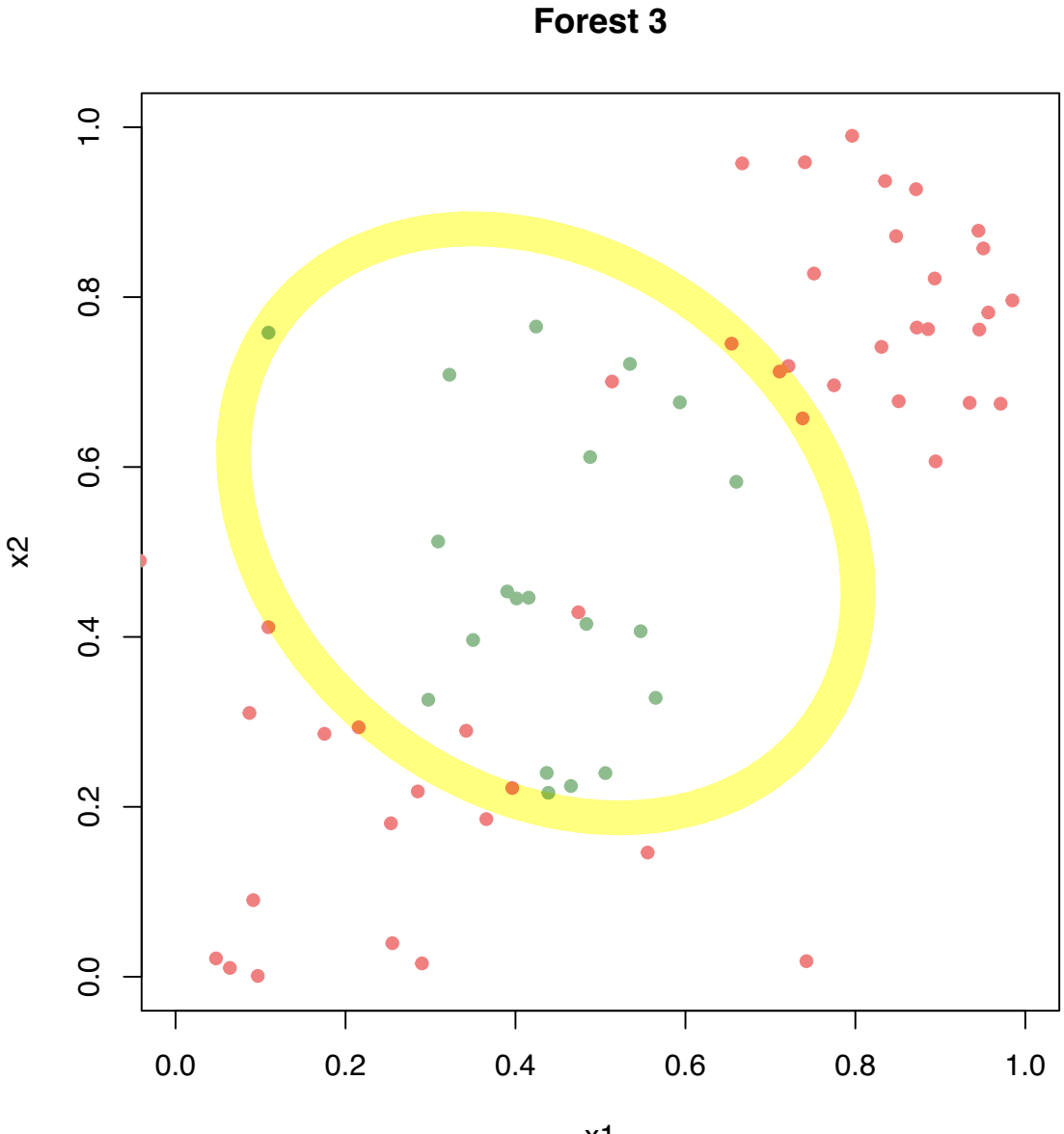
- seperable data
- linear boundary

You are also going to learn how you can find an optimal separating hyperplane when your data cannot be perfectly separated by a straight line, as in Forest 2. This leads to a classifier called a **Support Vector Classifier** or a **Soft Margin Classifier**.



not separable but linear boundary will work well

The Support vector classifier can be generalised to an approach that produces non-linear decision boundaries. This is called the **Support Vector Machine** (SVM) and is useful when the data is distributed as illustrated in Forest 3.

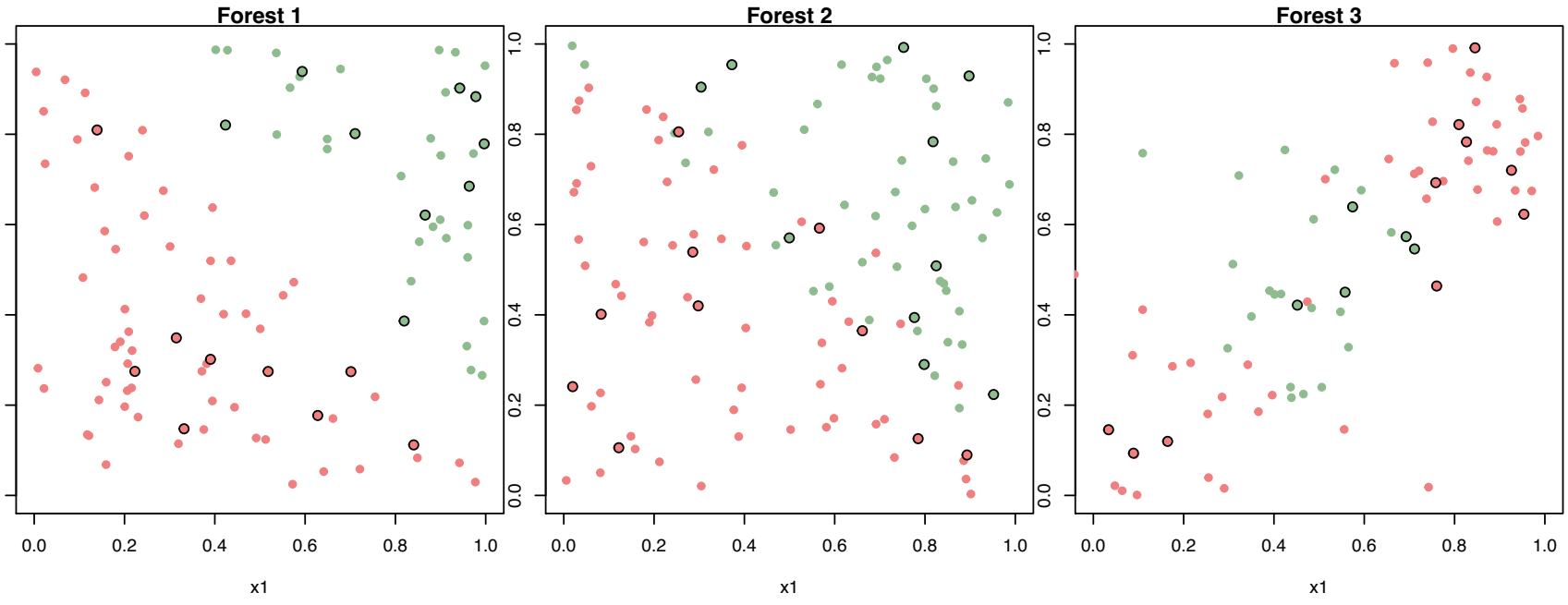


not seperable
linear boundary
will not work well

Test set & predictions

Code response $Y = \begin{matrix} -1 \\ +1 \end{matrix}$ not 0,1 as before

NB



black edges = test observations

Maximal Margin Classifier

Hyperplane

A **hyperplane** in p-dimensions is defined as

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = \beta_0 + \mathbf{x}^T \boldsymbol{\beta} = 0.$$

and is a $p - 1$ dimensional subspace.

- If a point $X = (X_1, X_2, ..., X_p)^T$ satisfies the above equation, it lies on the hyperplane.
- If $\beta_0 = 0$ the hyperplane goes through the origin (origo).
- The vector $\beta_1, \ldots, \beta_p$ (not including β_0) is called the normal vector and points in the direction orthogonal to the hyperplane.

Example of hyperplane $f(x_1, x_2) = x_1 + x_2 - 2 = 0$ or $f(x_1, x_2) = \frac{1}{\sqrt{2}} x_1 + \frac{1}{\sqrt{2}} x_2 - \sqrt{2} = 0$

(β_1, β_2, β_0)

If require $\beta_1^2 + \beta_2^2 = 1$

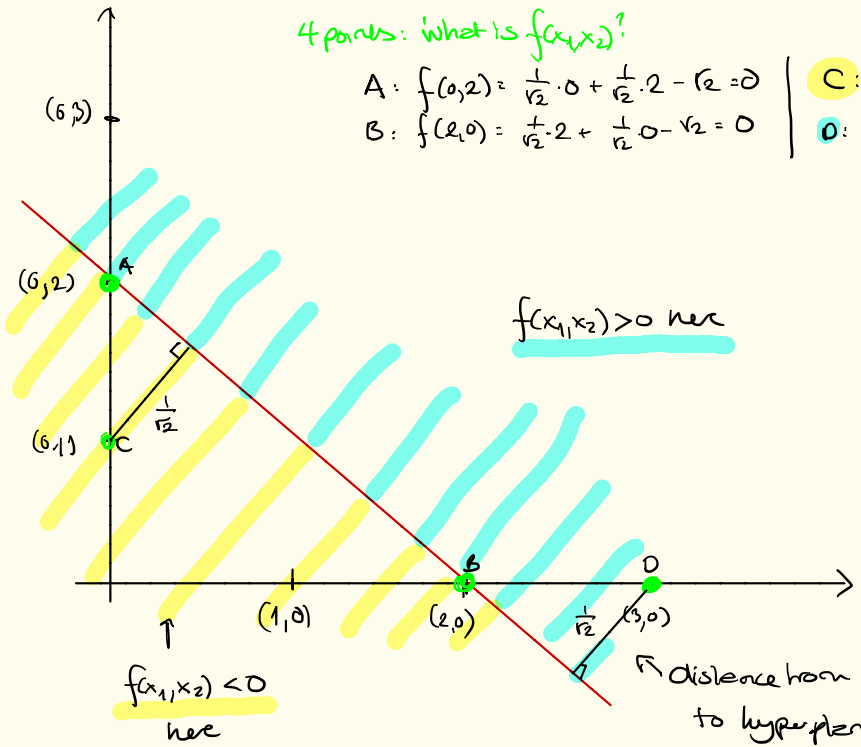
4 points: what is $f(x_1, x_2)$?

A: $f(0,2) = \frac{1}{\sqrt{2}} \cdot 0 + \frac{1}{\sqrt{2}} \cdot 2 - \sqrt{2} = 0$

B: $f(2,0) = \frac{1}{\sqrt{2}} \cdot 2 + \frac{1}{\sqrt{2}} \cdot 0 - \sqrt{2} = 0$

C: $f(0,1) = \frac{1}{\sqrt{2}} \cdot 0 + \frac{1}{\sqrt{2}} \cdot 1 - \sqrt{2} = -\frac{1}{\sqrt{2}}$ negative

D: $f(3,0) = \frac{1}{\sqrt{2}} \cdot 3 + \frac{1}{\sqrt{2}} \cdot 0 - \sqrt{2} = \frac{1}{\sqrt{2}}$ positive



$f(x_1, x_2) > 0$ here

$f(x_1, x_2) < 0$ here

↖ distance from point D to hyperplane is $|f(x_1, x_2)|$ when $\beta_1^2 + \beta_2^2 = 1$ (normalized)

If a point $X = (X_1, X_2, ..., X_p)^T$ satisfies

- $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = \beta_0 + \mathbf{x}^T \boldsymbol{\beta} > 0$ it lies on one side of the hyperplane, while if it satisfes
- $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p = \beta_0 + \mathbf{x}^T \boldsymbol{\beta} < 0$ it lies on the opposite side of the hyperplane.
- For normalized βs ($\sum_{i=1}^{p} \beta_j^2 = 1$) the value of $\beta_0 + \mathbf{x}^T \boldsymbol{\beta}$ gives the distance from the hyperplane.

Assumptions

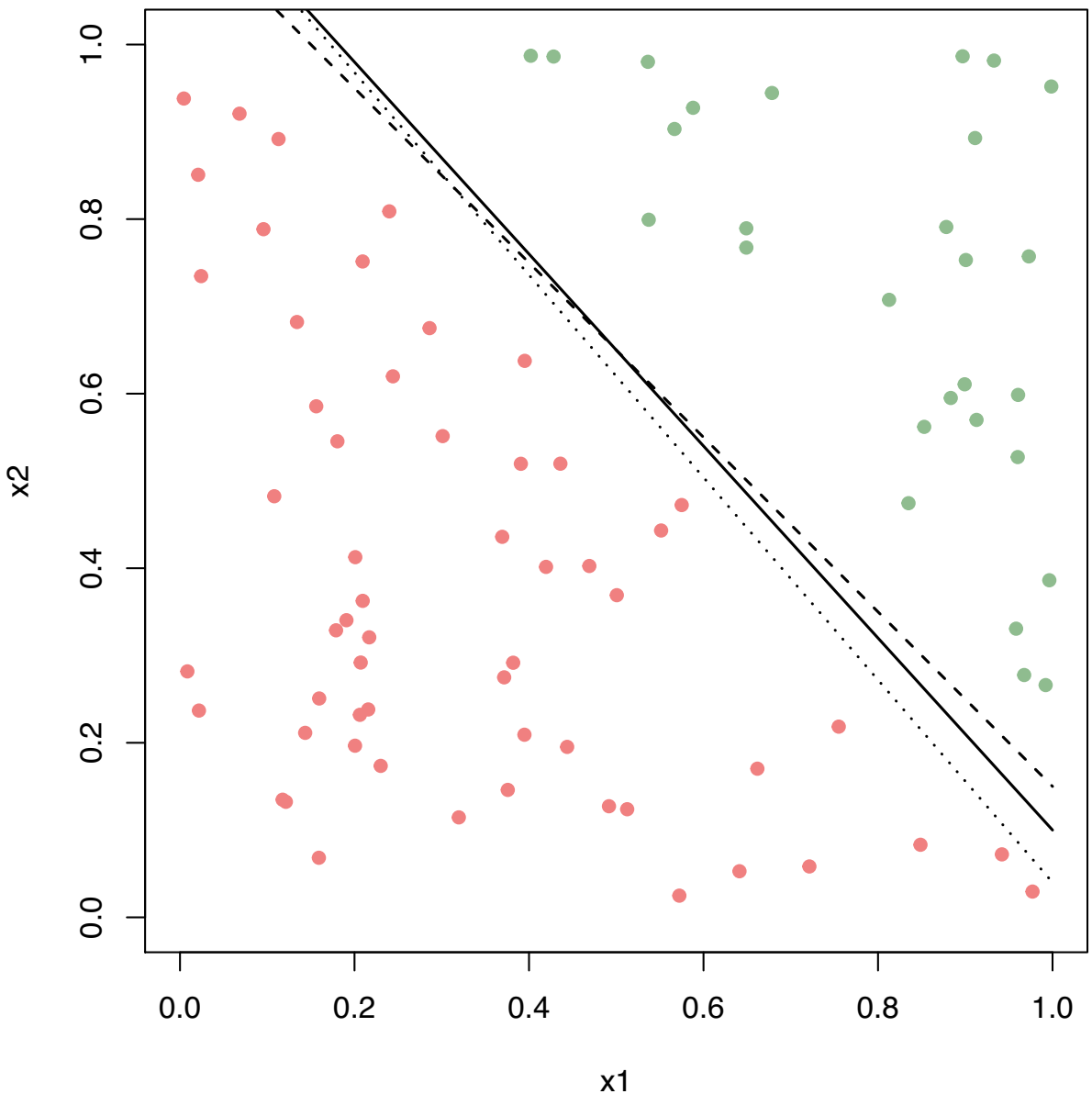
Assume that we have n training observations with p predictors

$$\mathbf{x}_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, \mathbf{x}_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$

not $\{0,1\}$

and that the responses $\boldsymbol{y}$ fall into two classes $y_1, \ldots, y_n \in \{-1, 1\}$. Further, assume that it is possible to separate the training observations perfectly according to their class.

(will see soon why)



many possible
seperating lines
(fence)
the
class
boundary

The three lines displayed in the figure are three possible separating hyperplanes for this dataset which contains two predictors x_1 and x_2 ($p = 2$). The hyperplanes have the property that

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} = \beta_0 + \mathbf{x}_i^T \boldsymbol{\beta} > 0$$

if $y_i = 1$ (green points) and

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} = \beta_0 + \mathbf{x}_i^T \boldsymbol{\beta} < 0$$

if $y_i = -1$ (orange points).

This means that for all observations (all are correctly classified)

$$y_i(\beta_0 + \mathbf{x}_i^T \beta) > 0$$

this is why we use $y \in \{-1, 1\}$

The hyperplane leads to a natural classifier:

We can assign a class to a new observation depending on which side of the hyperplane it is located. We denote the new observation $x^* = (x_1^*, ..., x_n^*)$ and classify it as $y^* = 1$ if

$$f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + ... + \beta_p x_p^* > 0$$

and as $y^* = -1$ otherwise.

The next question is which hyperplane we should choose.

In the above figure we plotted three possible hyperplanes, but there exist infinitely many possible separating hyperplanes for this dataset.

A natural choice is the **maximal margin hyperplane**. This is the separating hyperplane that is farthest from the training observations.

(From a statistical point of view we might be afraid that we are overfitting the data now.)

Optimization problem

- The maximal margin hyperplane is found by computing the perpendicular distance from each training observation to a given separating hyperplane.
- The smallest such distance is the minimal distance from the observations to the hyperplane, also known as the margin. (See illustration below.)
- We want to maximize this margin.

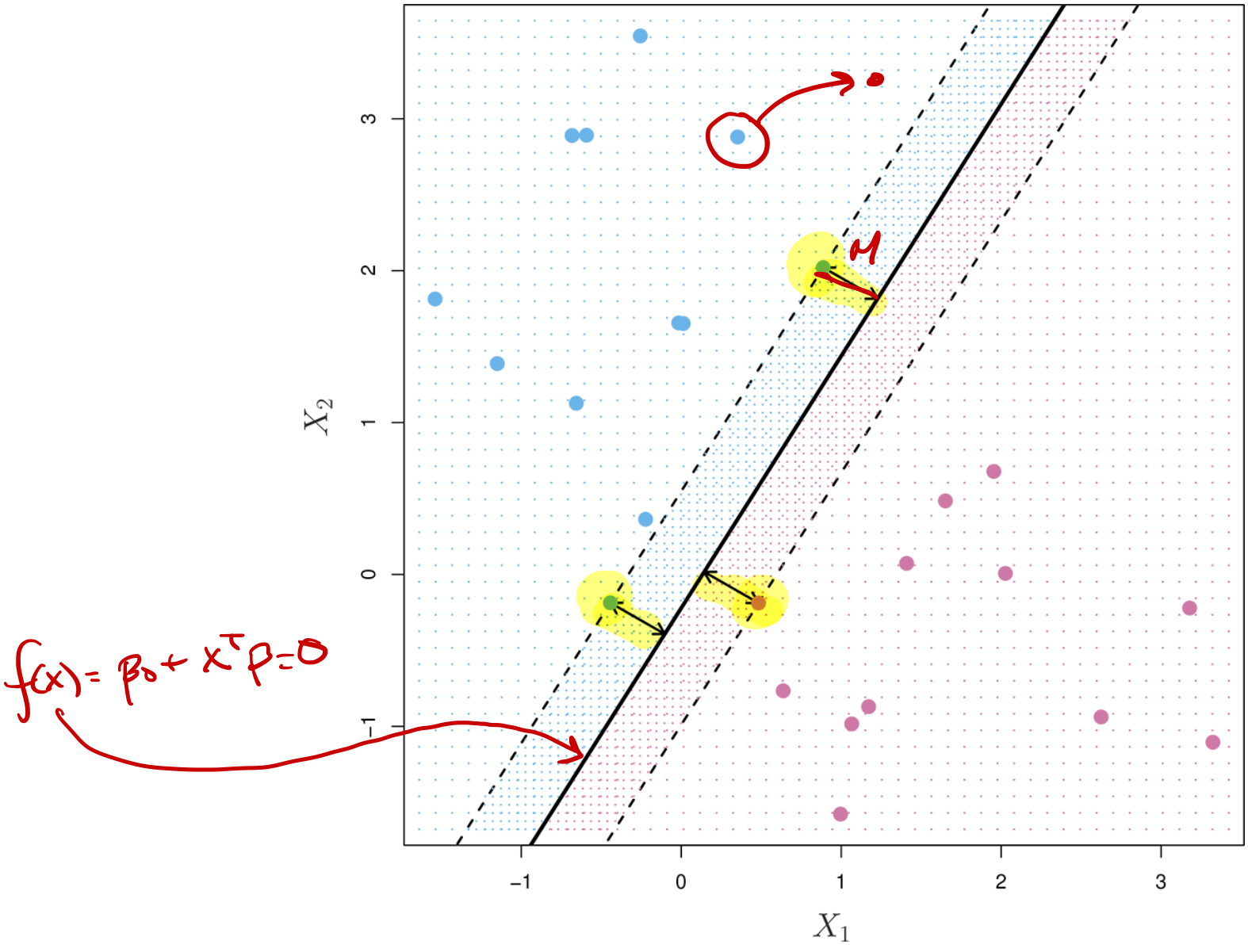


Figure 1: ISLR Figure 9.3

The process of finding the maximal margin hyperplane for a dataset with p covariates and n training observations can be formulated through the following optimization problem:

$$\text{maximize}_{\beta_0,\beta_1,\ldots,\beta_p} \quad M$$

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

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M \quad \forall i = 1, \ldots, n$$

where M is the width of the margin.

Observe: $y_i(\beta_0 + \mathbf{x}^T\beta)$ is the (signed) distance from the ith point to the hyperplane defined by the βs. We want to find the hyperplane, where each observaton is at least M units away - on the correct side, where M is as big as possible.

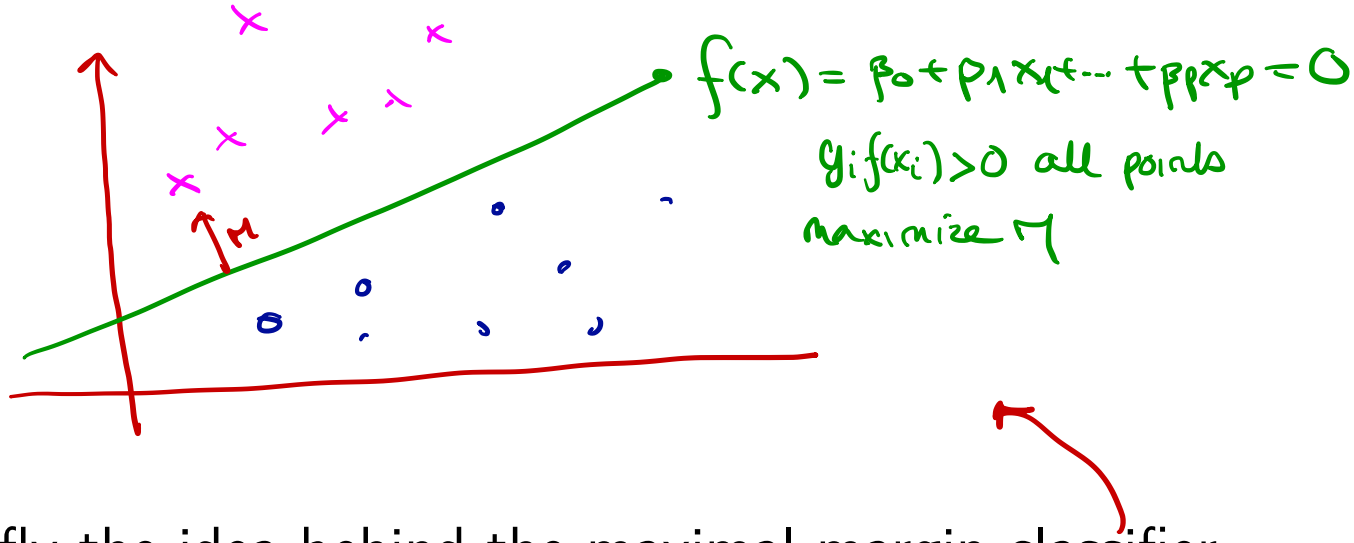
Above three of the observations are equidistant from the hyperplane. These are called **support vectors**. If one of the support vectors changes its position, the whole hyperplane will move. This is a property of the maximal margin hyperplane: It only depends on the support vectors, and not on the other observations.

It can be shown, see for example Efron and Hastie (2016) Section 19.1 and Friedman, Hastie, and Tibshirani (2001) Section 4.5, that the optimization problem can be reformulated using Lagrange multipliers (primal and dual problem) into a quadratic convex optimization problem that can be solved efficiently.

However, we do of cause have to solve the optimization problem to identify the support vectors and the unknown parameters for the separating hyperplane.

Since we in TMA4268 Statistical learning do not require a course in optimization - we do not go into details here.

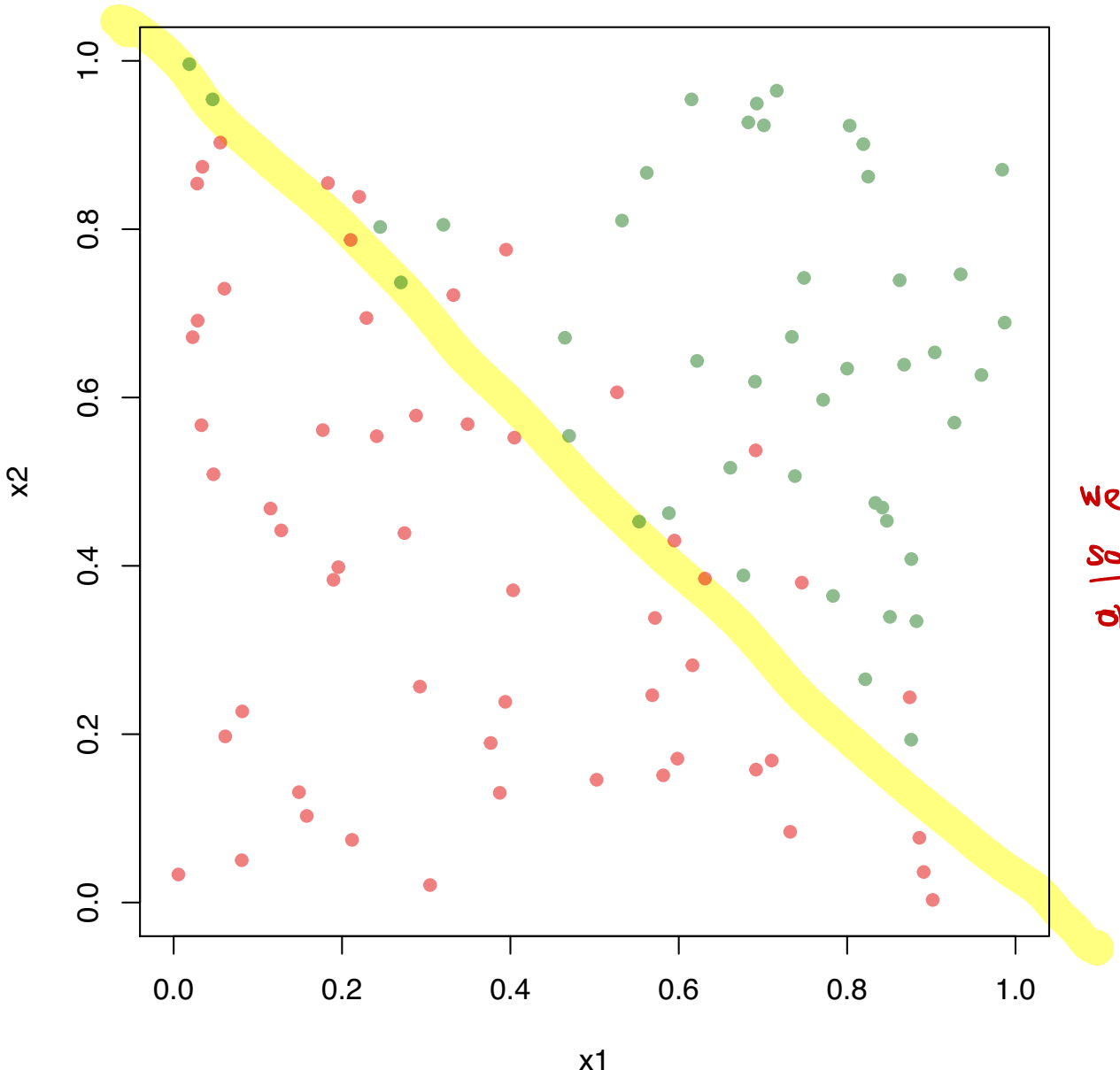
Questions



- Explain briefly the idea behind the maximal margin classifier.
- Is there any tuning parameters that need to be chosen? NO!
- What if our problem is not separable by a hyperplane? NEXT →

Support Vector Classifiers

For some data sets a separating hyperplane does not exist, it is *non-separable*.



We need to allow for some misclassified observations

Also, in some situation we what to allow for some misclassifications to make the class boundaries more robust to future observations - that is, we have noisy data or outliers are present.

In the special case where we have more predictors than observations it is possible to find a separating hyperplane, but the might not be the "best" hyperplane for us.

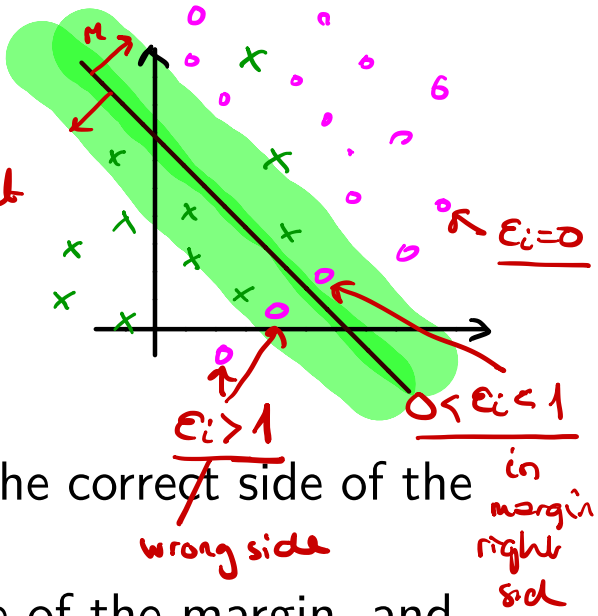
We now relax the maximal margin classifier to allow for a *soft-margin classifier*.

Optimization problem

$$\text{maximize}_{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n,} \; M \text{ subject to } \sum_{j=1}^{p} \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \quad \forall i = 1, \ldots, n.$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C.$$



- M is the width of the margin.
- $\epsilon_1, \ldots, \epsilon_n$ are *slack variables*.
  - If $\epsilon_i = 0$ it means that observation i is on the correct side of the margin,
  - if $\epsilon_i > 0$ observation i is on the wrong side of the margin, and
  - if $\epsilon_i > 1$ observation i is on the wrong side of the hyperplane.
- C is a *tuning (regularization) parameter* (chosen by cross-validation) giving the *budget for slacks*. It restricts the number of the training observations that can be on the wrong

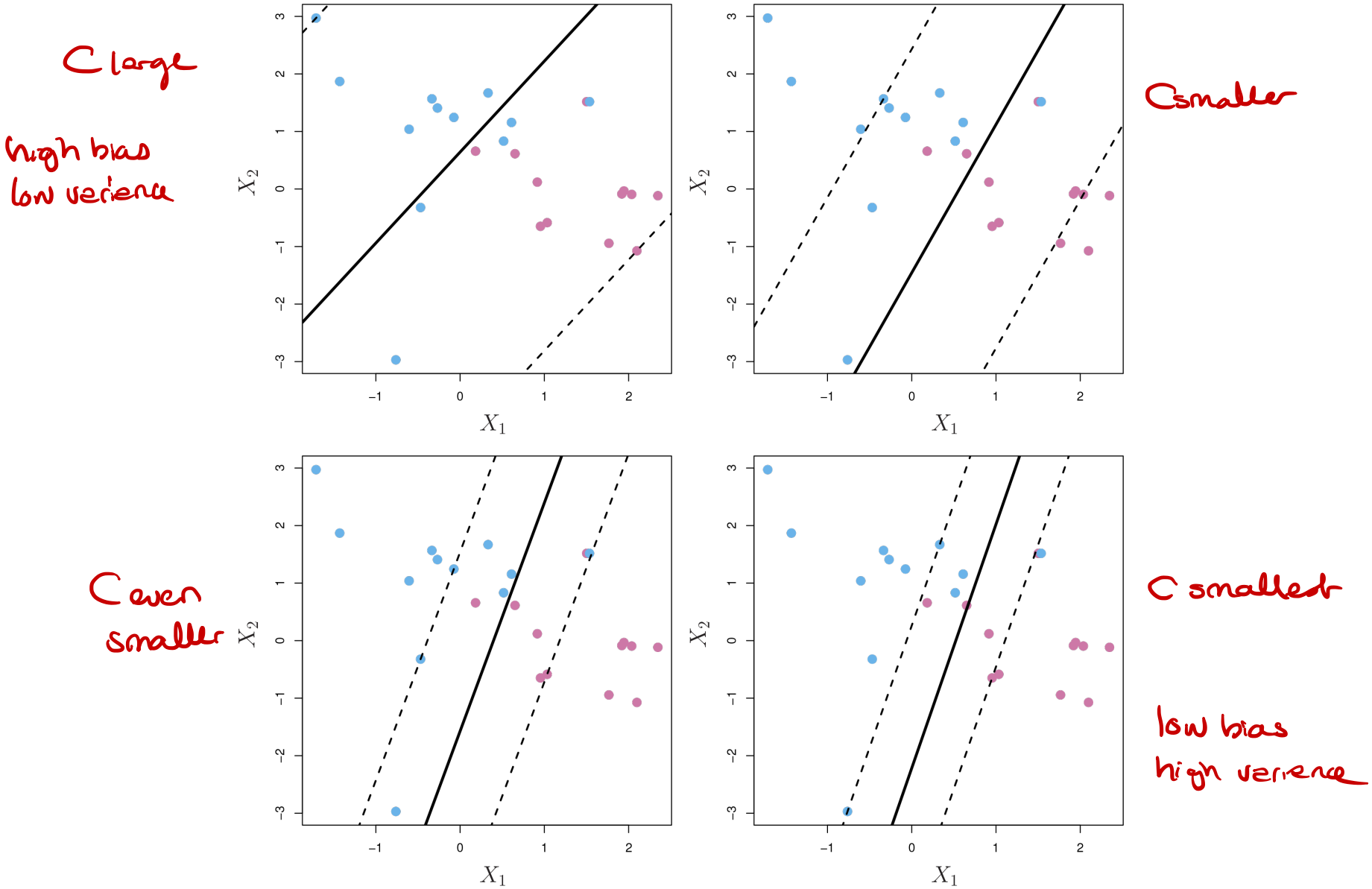


Figure 2: ISLR Figure 9.7: Top left=large C, smaller top right, bottom left and bottom right. As C decreases the tolerance for observations being on the wrong side of the margin decreases and the marin narrows.

The hyperplane has the property that it **only** depends on the observations that **either lie on the margin or on the wrong side of the margin**.

These observations are called our **support vectors**. The observations on the correct side of the margin do not affect the support vectors. The length of distance for the support vectors to the class boundary is proportional to the slacks.

Classification rule: We classify a test observation $\mathbf{x}^*$ based on the sign of $f(x^*) = \beta_0 + \beta_1 x_1^* + ... + \beta_p x_p^*$ as before:

- If $f(\mathbf{x}^*) < 0$ then $y^* = -1$.
- If $f(\mathbf{x}^*) > 0$ then $y^* = 1$.

More on solving the optimization problem: Friedman, Hastie, and Tibshirani (2001) Section 12.2.1 (primal and dual Lagrange problem, quadratic convex problem).

Questions

yes, same as for ridge and lasso

- Should the variables be standardized before used with this method?
- The support vector classifier only depends on the observations that violate the margin. How does C affect the width of the margin?

C large ⇒ accept many violations ⇒ M will be wide

- Discuss how the tuning parameter C affects the bias-variance trade-off of the method.

See previous page

Example

We will now find a support vector classifier for the second training dataset (forest2) and use this to classify the observations in the second test set (seeds2).

- There are 100 observations of trees: 45 pines ($y_i = 1$) and 55 redwood trees ($y_i = -1$).
- In the test set there are 20 seeds: 10 pine seeds and 10 redwood seeds.

The function `svm` in the package `e1071` is used to find the maximal margin hyperplane. The response needs to be coded as a factor variable, and the data set has to be stored as a dataframe.

```
library("e1071")
forest2=read.table(file="https://www.math.ntnu.no/emner/TMA4268/2018v/d
seeds2=read.table(file="https://www.math.ntnu.no/emner/TMA4268/2018v/da
train2=data.frame(x=forest2[,1:2], y=as.factor(forest2[,3]))
test2=data.frame(x=seeds2[,1:2], y=as.factor(seeds2[,3]))
```

for the budget for errors

The `svm` function uses a slightly different formulation from what we wrote above.

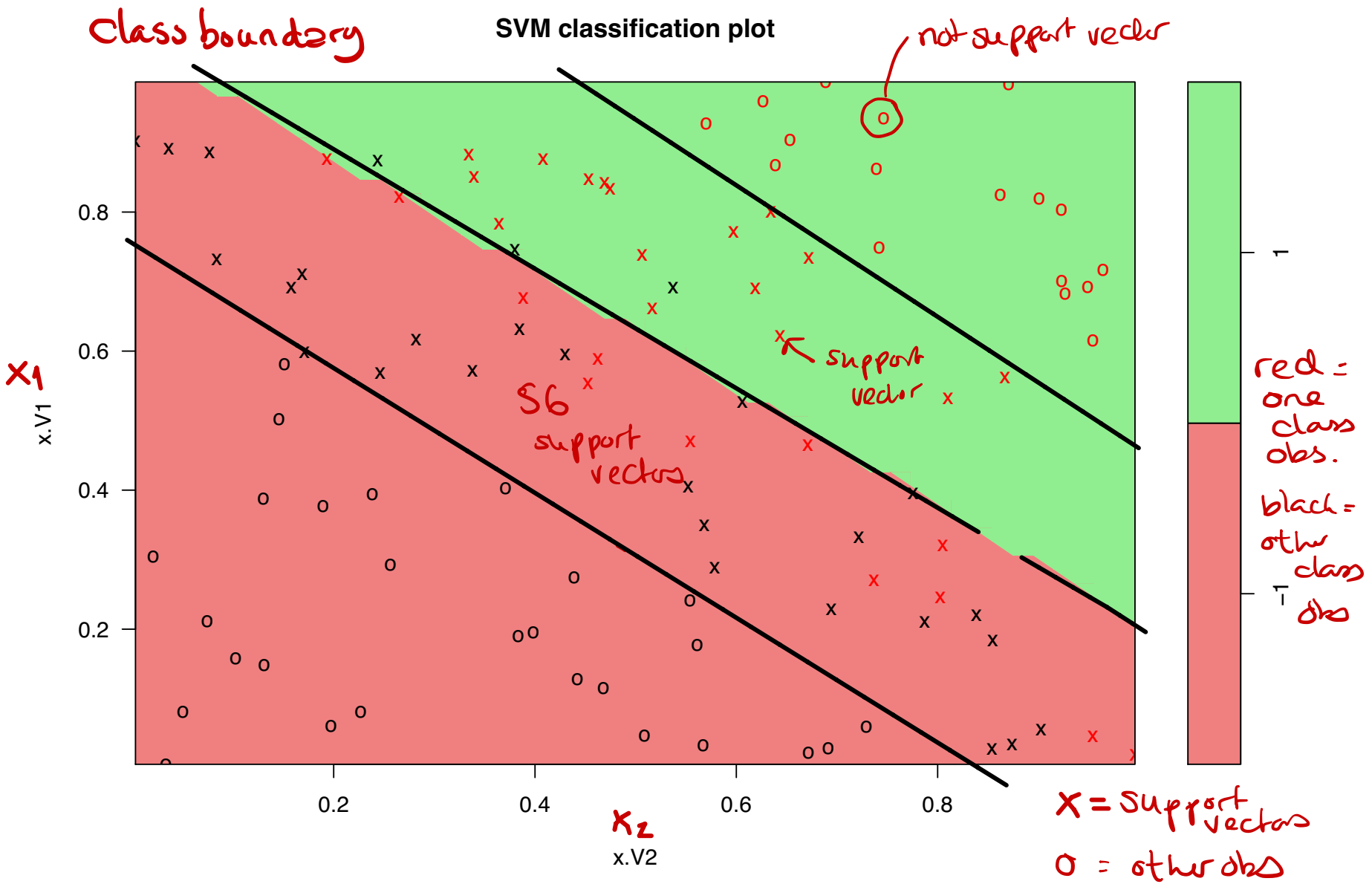
We had in our presentation a budget for errors C, but in `svm` we instead have an argument `cost` that allows us to specify the cost of violating the margin.

so - the opposite of budget

- When `cost` is set to a low value, the margin will be wider than if set to a large value.

We first try with `cost=1`. We set `kernel='linear'` as we are interested in a linear decision boundary. `scale=TRUE` scales the predictors to have mean 0 and standard deviation 1. We choose not to scale.

```r
svmfit_linear1=svm(y ~ ., data=train2, kernel='linear', cost=1, scale=F
plot(svmfit_linear1,train2,col=c("lightcoral","lightgreen"))
```



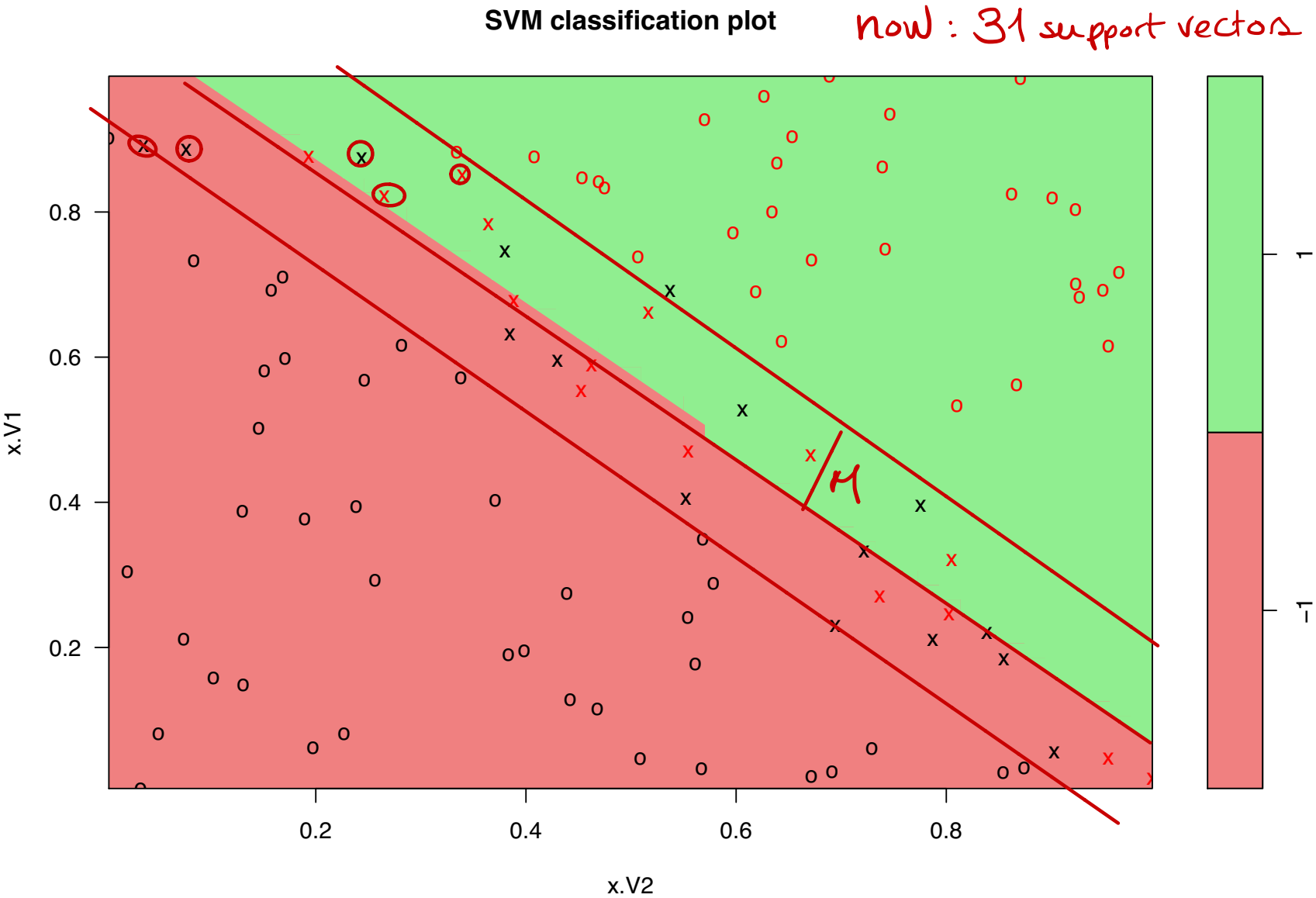

```r
summary(svmfit_linear1)
```

Observations

- Remark that the x_1 is plotted on the vertical axis, and the the implementation of the plotting function is made in a way that the linear boundary looks jagged.
- The crosses in the plot indicate the support vectors. With $cost = 1$, we have 56 support vectors, 28 in each class.
- All other observations are shown as circles.

Next, we set *cost* = 100: higher cost ⇒ narrower margin

```
svmfit_linear2=svm(y ~ ., data=train2, kernel='linear', cost=100, scale
plot(svmfit_linear2,train2,col=c("lightcoral","lightgreen"))
```



With *cost* = 100 we have 31 support vectors, i.e the width of the margin is decreased.

How do we find an optimal *cost* parameter? By using the *tune()* function we can perform ten-fold cross-validation and find the cost-parameter that gives the lowest cross-validation error:

```
set.seed(1)
CV_linear=tune(svm,y~.,data=train2,kernel="linear",ranges=list(cost=c(0
summary(CV_linear)
```

list of costs

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.15
##
## - Detailed performance results:
##    cost error dispersion
```

CV10 misclassification rate

According to the *tune()* function we should set the cost parameter to 0.1. The function also stores the best model obtained and we can access it as follows:

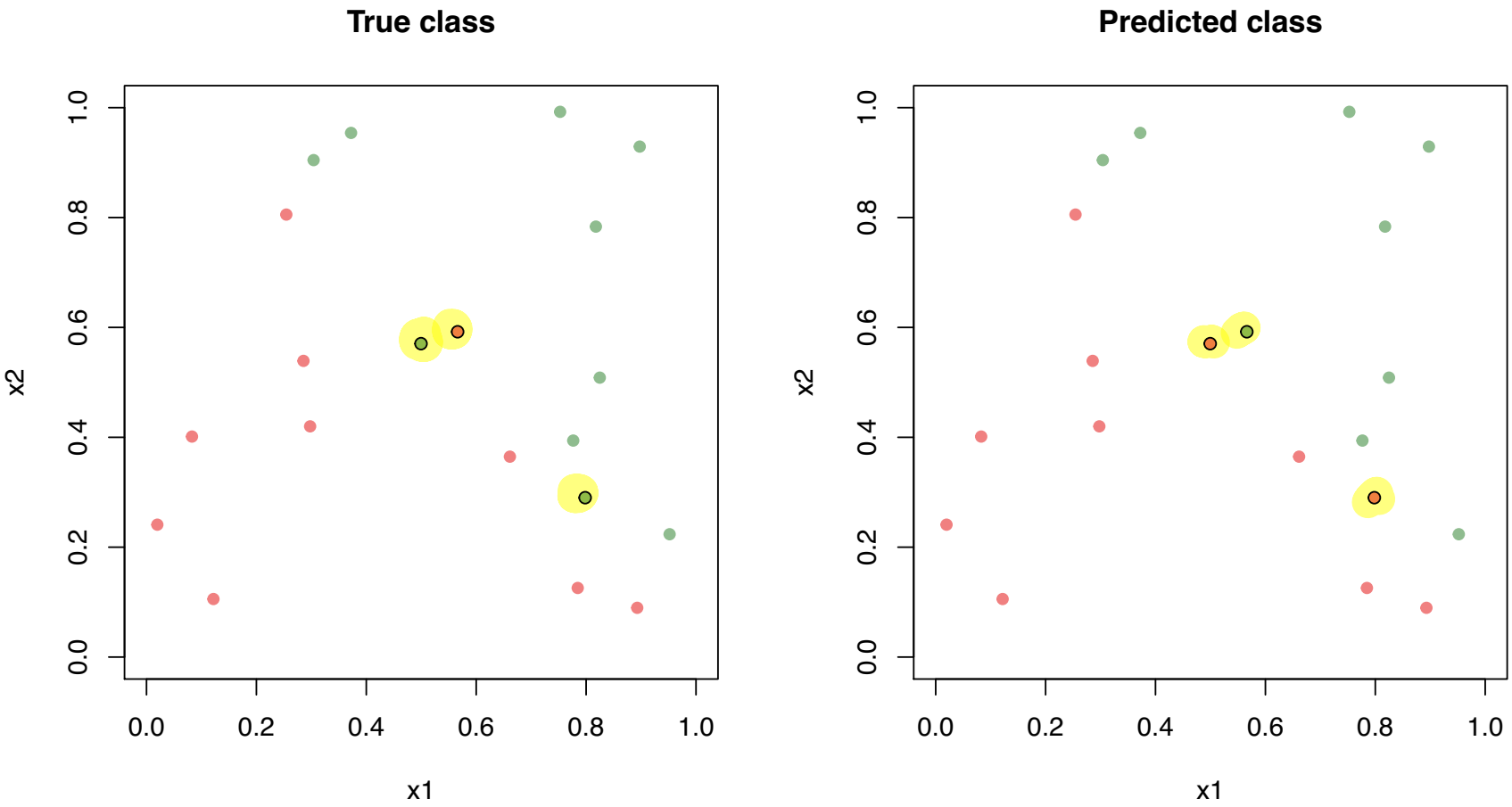
```
bestmod_linear=CV_linear$best.model
```

Next, we want to predict the class label of the seeds in the test set. We use the `predict` function and make a confusion table:

```
ypred_linear=predict(bestmod_linear,test2)
table(predict=ypred_linear,truth=test2[,3])
```

```
##         truth
## predict -1 1
##      -1  9 2
##      1   1 8
```

$\frac{3}{20} = 0.15$ also for test obs

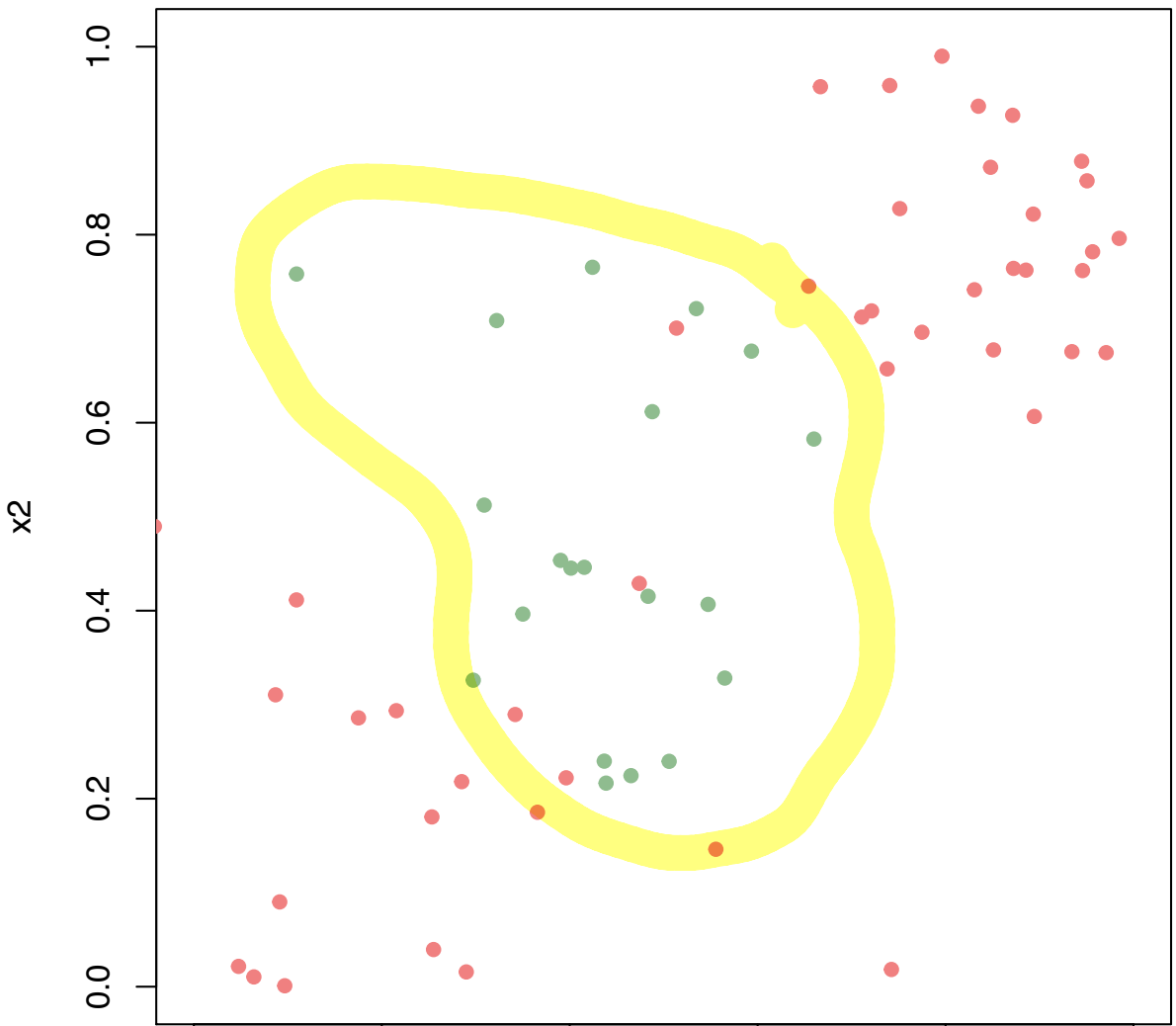

True class
Predicted class
x1
x2
0.0
0.2
0.4
0.6
0.8
1.0

In this case three of the test observations are misclassified: These three observations are marked with a black circle in the plot, and we observe that they lie on the border between the green and the orange points which is reasonable: The test observations located on the border between green and orange are hardest to predict.

Missing: the `svm` function is not (directly) outputting the equation for the class boundary, and not the value for the width of the margin. Want to see how to find this? Go to the recommended exercises.

Support Vector Machines

For some datasets a non-linear decicion boundary between the classes is more suitable than a linear decision boundary. In such cases you can use a **Support Vector Machine** (SVM). This is an extension of the support vector classifier.



$$f(x) = \beta_0 + \beta_1 \cdot x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \beta_4 \cdot x_2 + \beta_5 x_2^2 + \beta_6 \cdot x_2^3 + \beta_7 \cdot x_1 \cdot x_2^2 + \beta_8 \cdot x_1^2 \cdot x_2$$

Expanding the feature space

We saw in Module 7 that in regression we could fit non-linear curves by using a polynomial basis - adding polynomials of different order as covariates. This was a linear regression in the transformed variables, but non-linear in the original variables. Maybe we may add many such extra features and find a nice linear boundary in that high-dimensional space?

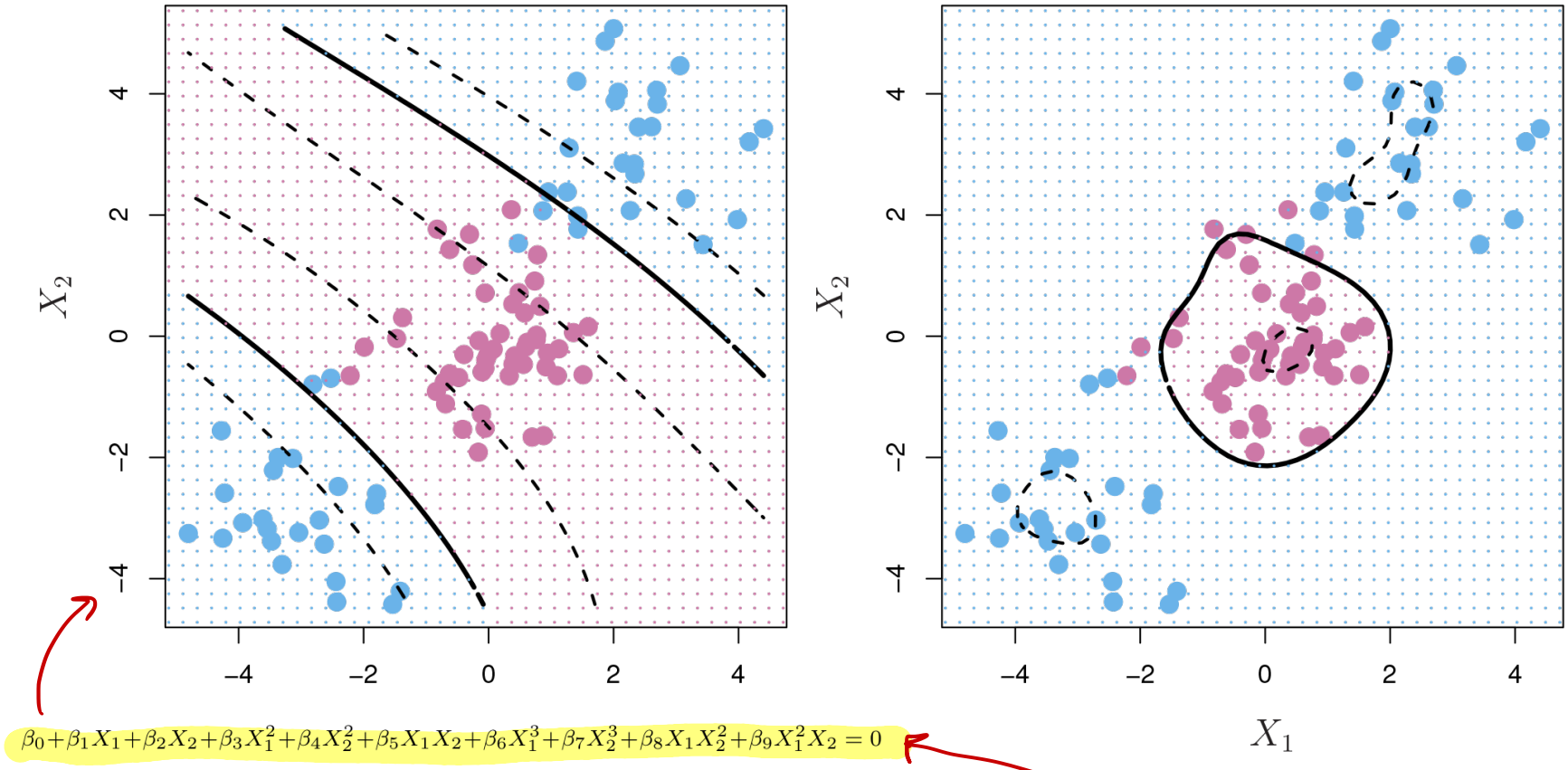


Figure 3: ISLR Figure 9.9

Left: expanding feature space to include cubic polynomials (9 parameters to estimate), and also observe the margins. (Right: radial basis function kernel - wait a bit.)

Next: replace polynomials with *kernels* for elegance and computational issues.

Inner products

From $f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p = \beta_0 + x^T \beta$ to

We have not focused on how to solve the optimisation problem of finding the support vector classifier hyperplane, because this is outside the scope of this course.

However, it can be *shown* that the solution to the support vector classifier problem can be expressed as

$$f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle$$

p-vector; x_i is training obs i; new point x; p-vector; use $\alpha_1, \ldots, \alpha_n$ instead of $\beta_1, \ldots, \beta_p$

where α_i is some parameter and $i = 1, \ldots, n$. The term $\langle x_i, x_{i'} \rangle$ denotes the inner product between two observations and is defined as:

$$\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}.$$

This means that we need to estimate n parameters instead of p (and for our expanded feature space then p might be larger than n). (For the interested reader: See Eq. 19.22 and 19.23 of Efron and Hastie (2016).)

Further, it then turns out that to estimate the parameters $\beta_0, \alpha_1, ..., \alpha_n$ this can be based on the $\binom{n}{2}$ inner products $\langle x_i, x_i' \rangle$ between all pair of training observations.

Also, $\alpha_i = 0$ for the observations i that are *not* the support vectors. Thus, we only need the inner product between the training observations and the observations corresponding to support vectors, and

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

where $\mathcal{S}$ contains the indices of the support points. So, we have sparsity in the observations.

Remark: we could alternatively say that $\alpha_i = 0$ define the support vectors.

Q: Find the support vectors

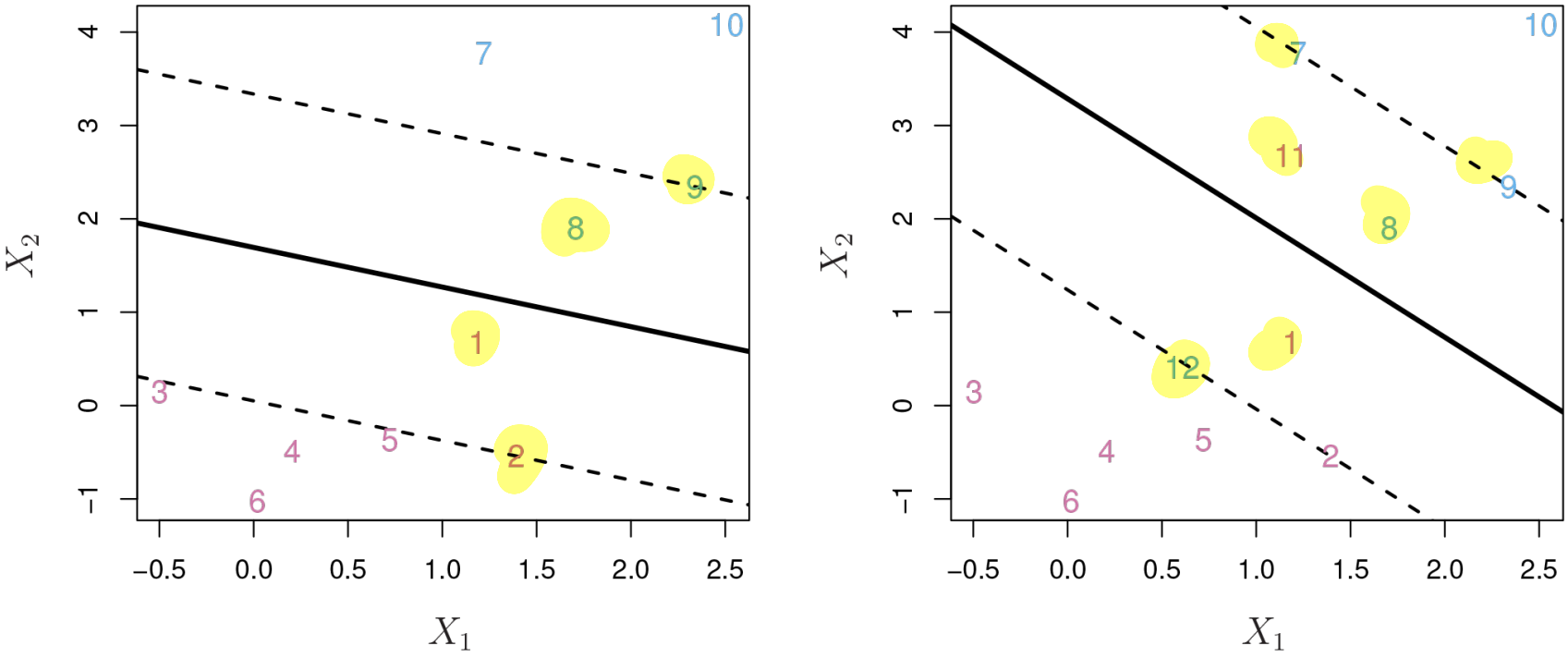


Figure 4: ISLR Figure 9.6

Observe: we need all observations to find the support vectors.

Kernels

The next step is now to *replace the inner product* $\langle x, x_i \rangle$ with a function $K(x_i, x_{i'})$ referred to as the **kernel**:

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

For the linear case (which is what we have considered so far), the kernel is simply the inner product $K(x_i, x_i') = \sum_{j=1}^{p} x_{ij} x_{i'j}$.

The two arguments to the kernel are two p-vectors.

If we want a more flexible decision boundary we could instead use a **polynomial kernel**. This polynomial kernel of degree $d > 1$ is given by:

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

(This kernel is not so much used in practice, but is popular for proofs.)

Using these kernels our solution for the class boundary can be written

estimate

$$\hat{f}(\mathbf{x}) = \beta_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i K(\mathbf{x}, \mathbf{x}_i)$$

The nice thing here is that we only need to calculate the kernels, not the basis functions (what we in Modul 7 did as extra columns of the design matrix).

$$f(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = \beta_0 + \sum_{i \in S} \alpha_i \cdot \underbrace{\langle x, x_i \rangle}_{K(x, x_i)}$$

or other nonlinear versions

A very popular choice is the radial kernel,

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

where γ is a positive constant (a tuning parameter).

Observe the connection to a multivariate normal density. If γ is small the decision boundaries are smoother than for larger γ.

It turns out that this computes the inner product in a very high (infinite) dimensional feature space. But, this does not give overfitting because some of the dimensions are "squashed down".

The radial kernel is convinient if we want a circular decision boundary, and γ can be chosen by cross-validation.

Study Figures 19.5 and 19.6 from Efron and Hastie (2016) to see how the radial kernel can make smooth functions.

Computer Age Statistical Inference ← press link

Kernels and our optimization

We now merge our optimization problem (from our support vector classifier) with our kernel representation $f(x)$ to get the Support Vector Machine (SVM).

$$\text{maximize}_{\beta_0,\alpha_1,\ldots,\alpha_n,\epsilon_1,\ldots,\epsilon_n,} \; M$$

$$y_i(f(\mathbf{x}_i)) \geq M(1-\epsilon_i) \quad \forall i = 1,\ldots,n.$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C.$$

where

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

Regularization parameter example

Heart data - predict heart disease from $p = 13$ predictors.

Training errors as ROC and AUC.

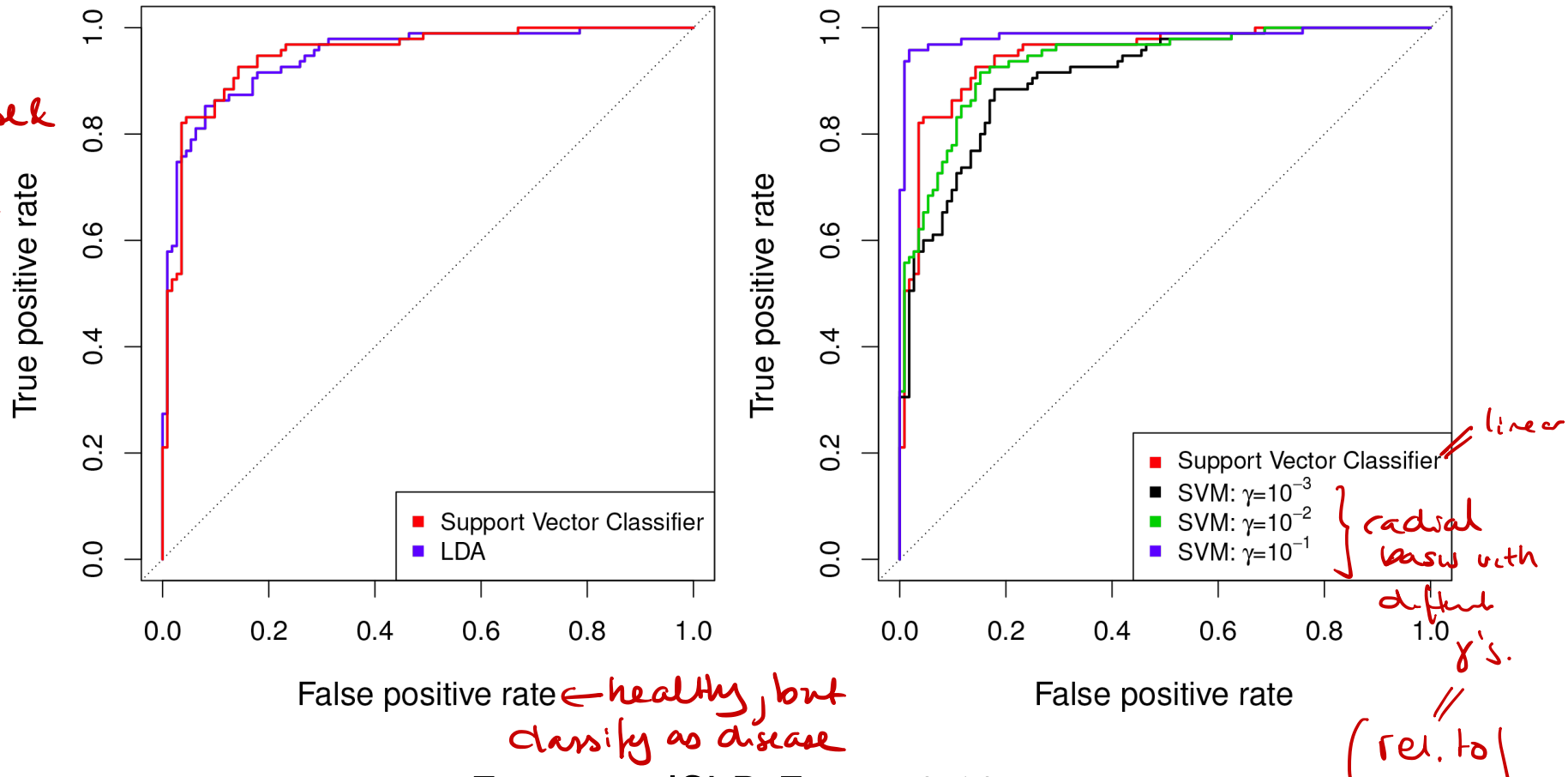


Figure 5: ISLR Figure 9.10

Heart data - test error.

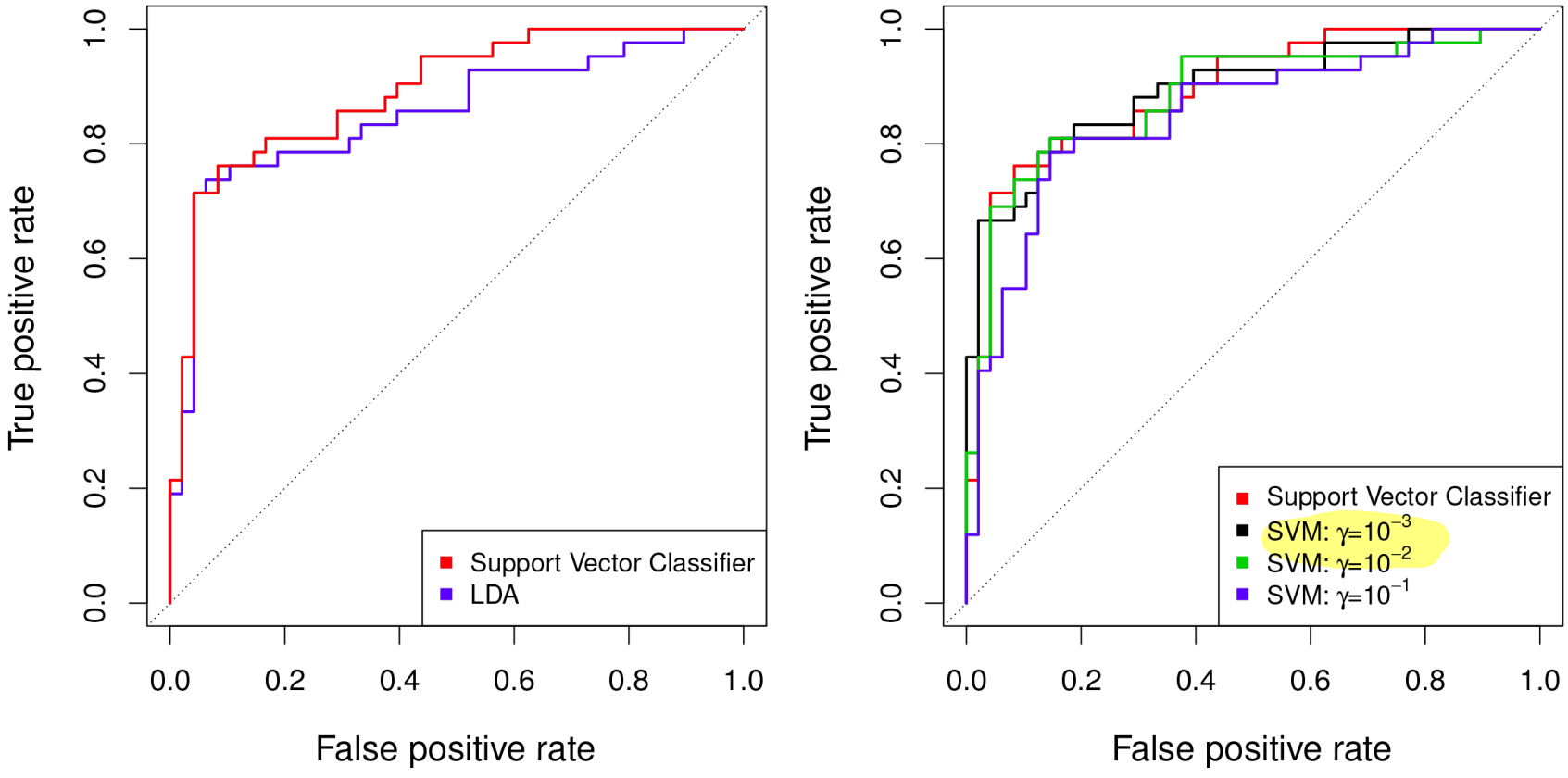


Figure 6: ISLR Figure 9.11

Example: forest 3

To illustrate the SVM we use the third training dataset (forest3) and the third test set (seeds3). We use the `svm`function as before. However, we now set `kernel='radial'` as we want a non-linear decision boundary:

See html-version of module

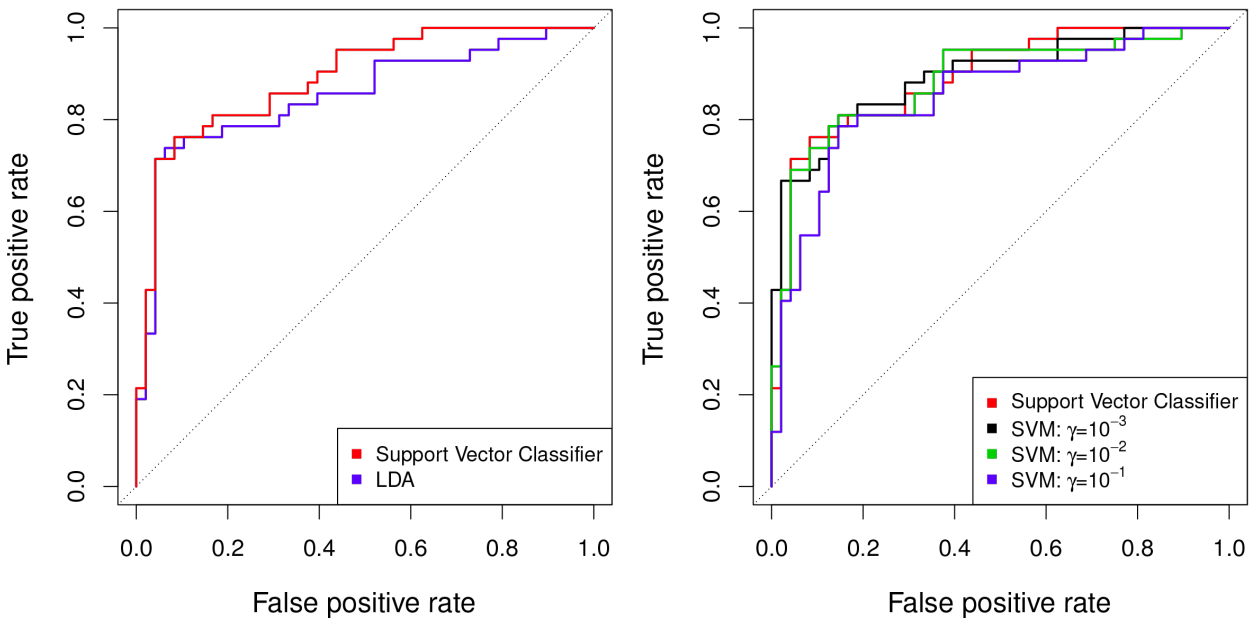


Figure 6: ISLR Figure 9.11

Heart data - test error.

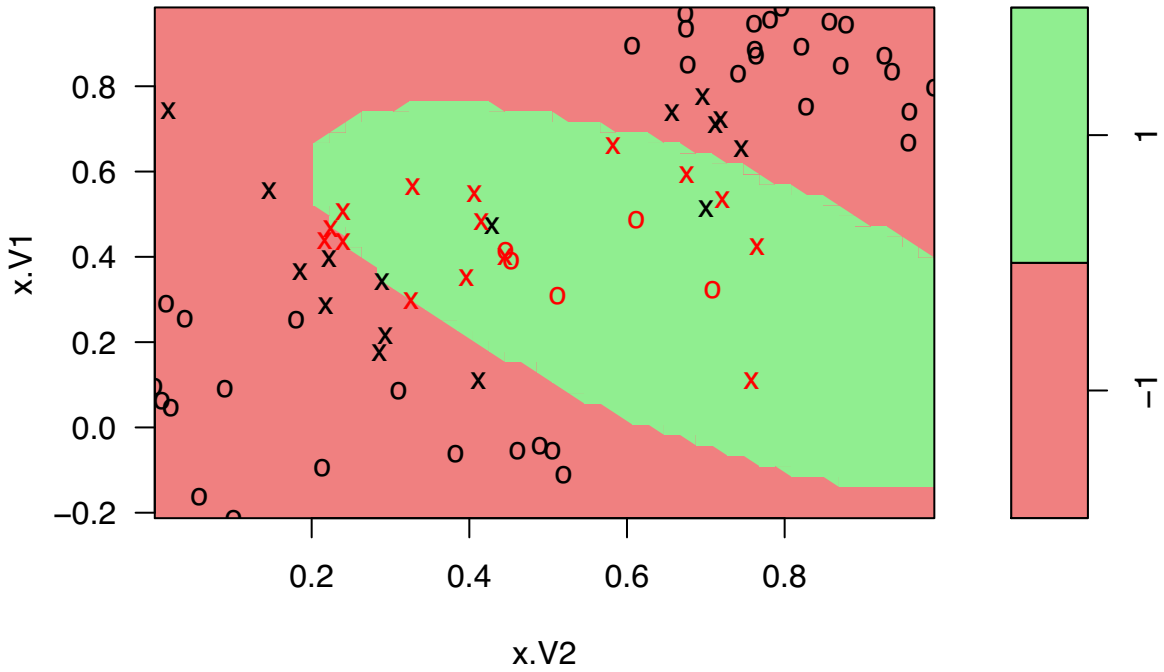
Example: forest 3

To illustrate the SVM we use the third training dataset (forest3) and the third test set (seeds3). We use the `svm`function as before. However, we now set `kernel='radial'` as we want a non-linear decision boundary:

```
library("e1071")
forest3=read.table(file="forest3.txt");
seeds3=read.table(file="seeds3.txt")

train3=data.frame(x=forest3[,1:2], y=as.factor(forest3[,3]))
test3=data.frame(x=seeds3[,1:2], y=as.factor(seeds3[,3]))

svmfit_kernel1=svm(y ~ ., data=train3, kernel='radial', gamma=1, cost=10, scale=FALSE)
plot(svmfit_kernel1,train3,col=c("lightcoral","lightgreen"))
```




```
summary(svmfit_kernel1)
```

```
##
## Call:
## svm(formula = y ~ ., data = train3, kernel = "radial", gamma = 1,
##     cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  1
##
## Number of Support Vectors:  31
##
##  ( 16 15 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

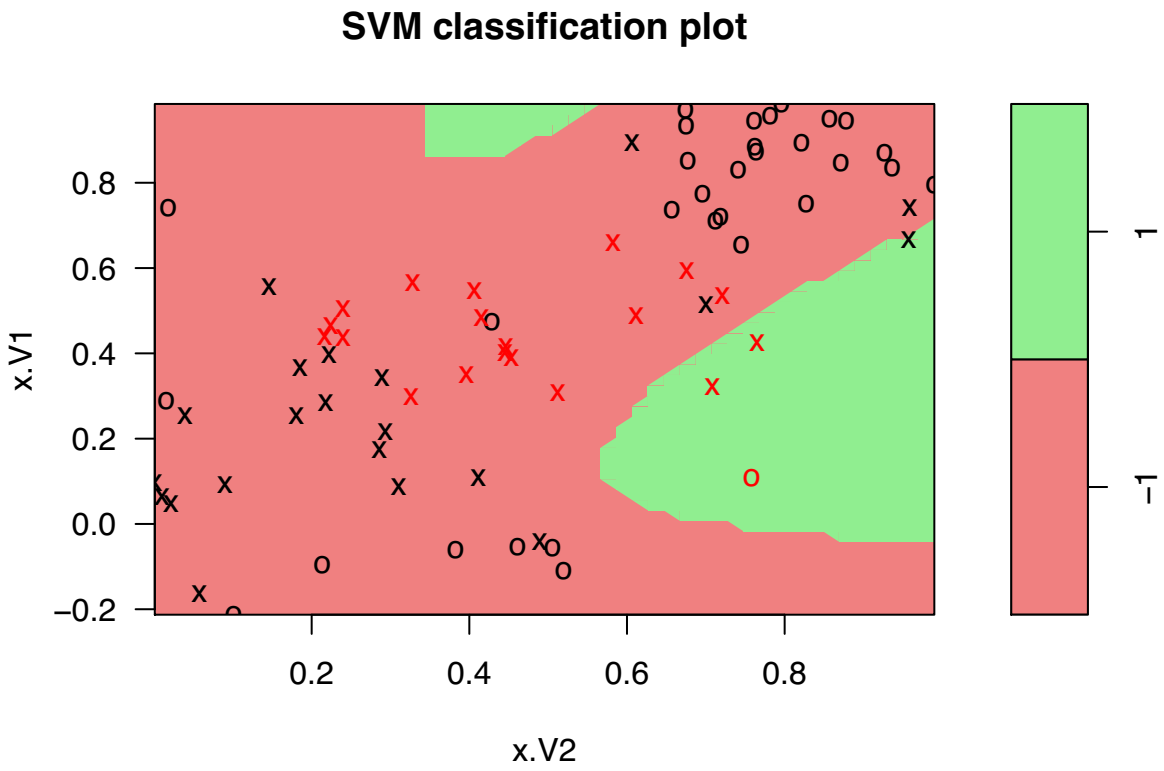
We could also try with a polynomial kernal with degree 4 as follows:

```
library("e1071")
forest3=read.table(file="forest3.txt");
seeds3=read.table(file="seeds3.txt")

train3=data.frame(x=forest3[,1:2], y=as.factor(forest3[,3]))
test3=data.frame(x=seeds3[,1:2], y=as.factor(seeds3[,3]))

svmfit_kernel2=svm(y ~ ., data=train3, kernel='polynomial', degree=4, cost=100000, scale=FALSE)
```

```
plot(svmfit_kernel2,train3,col=c("lightcoral","lightgreen"))
```




```
summary(svmfit_kernel2)
```

```
##
## Call:
## svm(formula = y ~ ., data = train3, kernel = "polynomial", degree = 4,
##     cost = 1e+05, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1e+05
##      degree:  4
##       gamma:  0.5
##      coef.0:  0
##
## Number of Support Vectors:  40
##
##  ( 21 19 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

For this dataset a radial kernal is a natural choice: A circular decision boundary seems like a good idea. Thus,

we proceed with `kernel='radial'`, and use the `tune()` function to find the optimal tuning parameter C:

```
set.seed(1)
CV_kernel=tune(svm,y~.,data=train3,kernel="radial",gamma=1,ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)
summary(CV_kernel)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1232143
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.2732143  0.1619332
## 2 1e-02 0.2732143  0.1619332
## 3 1e-01 0.2732143  0.1619332
## 4 1e+00 0.1357143  0.1268849
## 5 5e+00 0.1357143  0.1436486
## 6 1e+01 0.1232143  0.1379232
## 7 1e+02 0.1250000  0.1248582
```

The optimal C is 10. Next, we predict the class label of the seeds in the test set with a model with C=10, make a confusion table and plot the results:

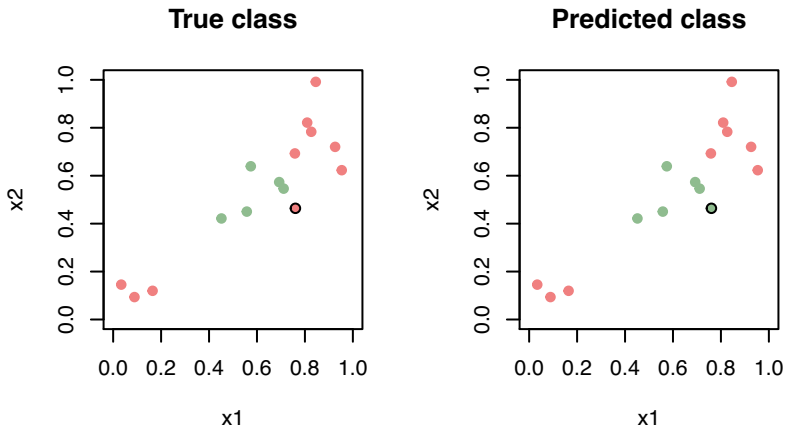
```
bestmod_kernel=CV_kernel$best.model
ypred_kernel=predict(bestmod_kernel,test3)

par(mfrow=c(1,3)); par(pty="s")
plot(NA,xlab="x1",ylab="x2",xlim=c(0,1),ylim=c(0,1));title("True class")
points(seeds3[seeds3[,3]==-1,1:2],pch=19,col="lightcoral",cex=0.9)
points(seeds3[seeds3[,3]==1,1:2],pch=19,col="darkseagreen",cex=0.9)
points(seeds3[which(ypred_kernel!=seeds3[,3]),1:2],pch=21) #Mark misclassification.


plot(NA,xlab="x1",ylab="x2",xlim=c(0,1),ylim=c(0,1));title("Predicted class")
points(seeds3[ypred_kernel==-1,1:2],pch=19,col="lightcoral",cex=0.9)
points(seeds3[ypred_kernel==1,1:2],pch=19,col="darkseagreen",cex=0.9)
points(seeds3[which(ypred_kernel!=seeds3[,3]),1:2],pch=21) #Mark misclassification.

table(predict=ypred_kernel,truth=test3[,3])
```

```
##        truth
## predict -1 1
##      -1  9 0
##      1   1 5
```



Only one seed is misclassified.

Extensions

More than two classes

What if we have k classes?

- OVA: one-versus-all. Fit k different two-class SVMs $f_k(x)$ where one class is compared to all other classes. Classify a test observation to the class where $f_k(x*)$ is largest.
- OVO: For multiclass-classification with k levels, k>2, libsvm uses the 'one-against-one'-approach, in which k(k-1)/2 binary classifiers are trained; the appropriate class is found by a voting scheme (the class that wins the most pairwise competitions are chosen).

Comparisons

Focus is comparing the support vector classifier and logistic regression

It is possible to write the optimization problem for the support vector classifier as a "loss"+"penalty":

$$\text{minimize}_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{n} \max(0, 1 - y_i f(\mathbf{x}_i)) + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

- the loss is called *hinge loss* - observe the max and 0 to explain why only support vectors contribute
- the penalty is a ridge penalty
- large λ gives βs small and more violations=high bias, but low variance
- small λ gives βs large and less violations=low bias, but high variance

---

Extensions

Ⓐ : 1 vs 2-K
2 vs 1, 3-K
3 vs 1,2,4-K
⋮
K vs 1,.., K-1
} K models

More than two classes

What if we have k classes?

Ⓐ

- OVA: one-versus-all. Fit k different two-class SVMs $f_k(x)$ where one class is compared to all other classes. Classify a test observation to the class where $f_k(x*)$ is largest.

Ⓑ

- OVO: For multiclass-classification with k levels, k>2, libsvm uses the 'one-against-one'-approach, in which k(k-1)/2 binary classifiers are trained; the appropriate class is found by a voting scheme (the class that wins the most pairwise competitions are chosen).

Ⓑ : 1vs2
1vs3
1vs4
⋮
k-1 vs k
} $\underbrace{\frac{k\cdot(k-1)}{2}}_{\binom{k}{2}}$ models

Comparisons

Focus is comparing the support vector classifier and logistic regression

It is possible to write the optimization problem for the support vector classifier as a "loss"+"penalty":

$$\text{minimize}_\beta \left\{ \sum_{i=1}^{n} \max(0, 1 - y_i f(\mathbf{x}_i)) + \lambda \sum_{j=1}^{p} \beta_j^2 \right\}$$

- the loss is called *hinge loss* - observe the max and 0 to explain why only support vectors contribute
- the penalty is a ridge penalty
- large λ gives βs small and more violations=high bias, but low variance
- small λ gives βs large and less violations=low bias, but high variance

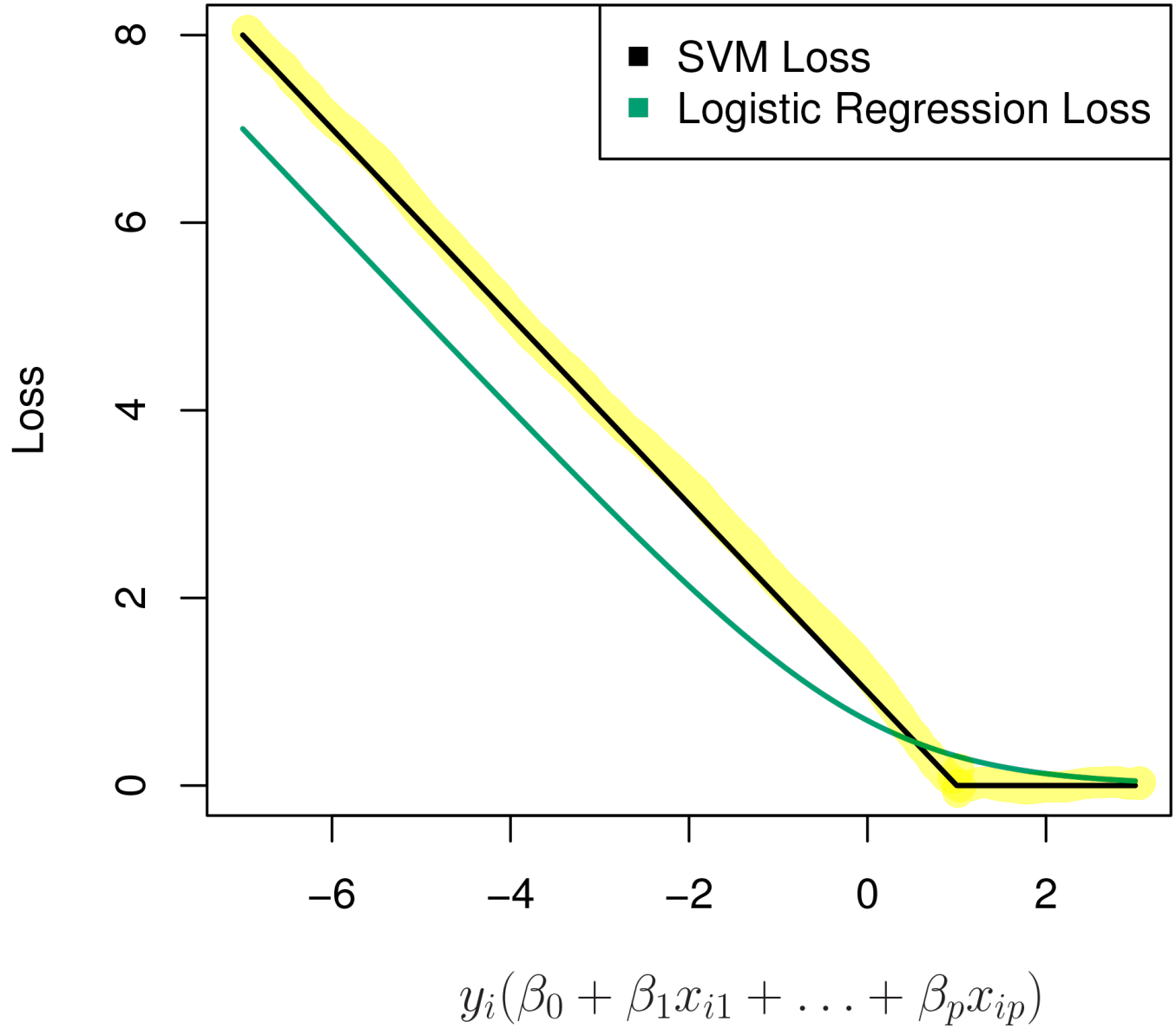


Figure 7: ISLR Figure 9.12: hinge loss - loss 0 for observations on the correct sideof the margin

Hinge loss:

$$\max(0, 1 - y_i f(x_i))$$

For comparison a logistic regression (with ridge penalty) would be (binomial deviance with -1,1 coding of y)

$$\log(1 + \exp(-y_i f(\mathbf{x}_i)))$$

It can be shown that in logistic regression all observations contribute weighted by $p_i(1 - p_i)$ (where p_i is probability for class 1), that fade smoothly with distance to the decision boundary

It is possible to extend the logistic regression to include non-linear terms, and ridge penalty,

When to use SVM?

- If classes are nearly separable SVM will perform better than logistic regression. (LDA will also perform better than logistic regression.)
- and if not, then a ridge penalty version of logistic regression are very similar to SVM, and logistic regression will also give you probabilities for each class.
- If class boundaries are non-linear then SVM is more popular, but kernel versions of logistic regression is possible, but more computationally expensive.

- We use methods from computer science, not probability models - but looks for a separating hyperplane in (an extended) feature space in the classification setting.
- SVM is a widely successful and a "must have tool"
- Interpretation of SVM: all features are included and maybe not so easy to interpret.
- Not so easy to get class probabilites from SVM (what is done is actually to fit a logistic regression after fitting SVM).

Recommended exercises

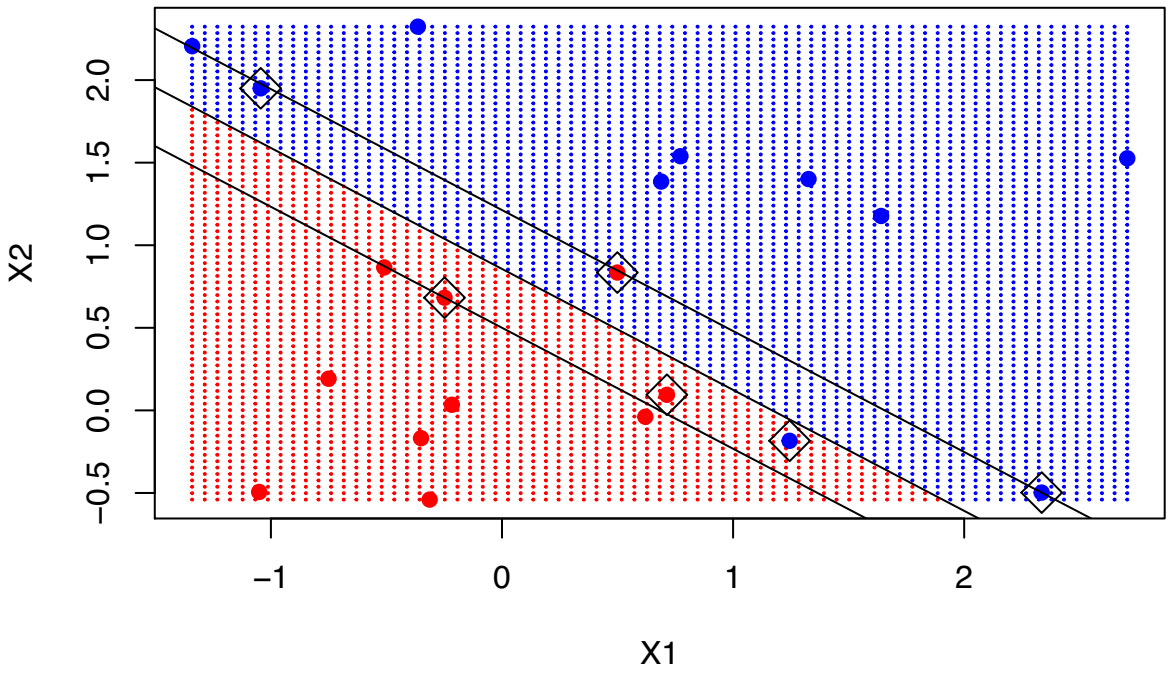
1. Understanding the algorithms:

- Exercise 1, 2 and 3 in the book.

2. Data analysis

- Go back and read in the `forest1` data (is located in the same place as `forest2`) and run the `svm` with a very high value for `cost`. The `forest1` is a separable problem.
- Linear version of SVM: Making nicer plots for SVM from Lab video. Go through the code and see what is happening (and see the video if you want more explanation).

```
# code taken from video by Trevor Hastie linked above
library(e1071)
# fake data
set.seed(10111)
x=matrix(rnorm(40),20,2)
```



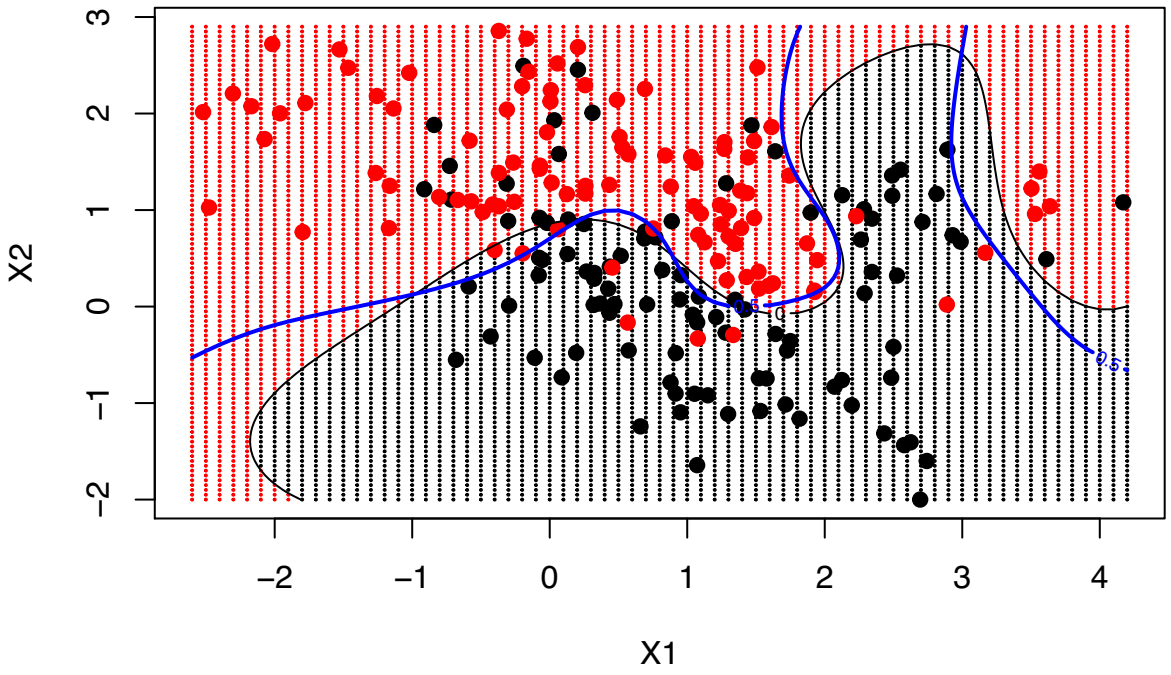
- SVM for non-linear class boundary using simulated data set from Friedman, Hastie, and Tibshirani (2001) where the truth is known (mixtures of normals probably).

```
load(url("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda"))
names(ESL.mixture)
```

```
## [1] "x"        "y"        "xnew"     "prob"     "marginal" "px1"
## [7] "px2"      "means"
```

```
rm(x,y)
attach(ESL.mixture)

plot(x,col=y+1)
```



R packages

These packages needs to be install before knitting this R Markdown file.

```
install.packages("e1071")
```

References

- Videoes on YouTube by the authors of ISL, Chapter 9, and corresponding slides
- Solutions to exercises in the book, chapter 9

Efron, Bradley, and Trevor Hastie. 2016. *Computer Age Statistical Inference - Algorithms, Evidence, and Data Science*. Cambridge University Press.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer series in statistics New York.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

Karatzoglou, A., D. Meyer, and K. Hornik. 2006. "Support Vector Machines in R." *Journal of Statistical Software* 15 (9).

R packages

These packages needs to be install before knitting this R Markdown file.

```
install.packages("e1071")
```

References

- Videoes on YouTube by the authors of ISL, Chapter 9, and corresponding slides
- Solutions to exercises in the book, chapter 9

Efron, Bradley, and Trevor Hastie. 2016. *Computer Age Statistical Inference - Algorithms, Evidence, and Data Science*. Cambridge University Press.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2001. *The Elements of Statistical Learning*. Vol. 1. Springer series in statistics New York.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 112. Springer.

Karatzoglou, A., D. Meyer, and K. Hornik. 2006. "Support Vector Machines in R." *Journal of Statistical Software* 15 (9).