Module 9: Support Vector Machines

TMA4268 Statistical Learning V2020

Stefanie Muff, Department of Mathematical Sciences, NTNU

March 6 and 9, 2020

Last update: March 9, 2020

Acknowledgements

- A lot of this material stems from Mette Langaas and her TAs (in particular Thea Roksvåg, who developed the set of slides, but also Mette Langaas and Julia Debik). Thanks to Mette for the permission to use the material!
- Some of the figures and slides in this presentation are taken (or are inspired) from James et al. (2013).

Introduction

Learning material for this module

- James et al (2013): An Introduction to Statistical Learning. Chapter 9.
- All the material presented on these module slides and in class.
- Check also the "Further reading" slide at the end of these module slides.

What will you learn?

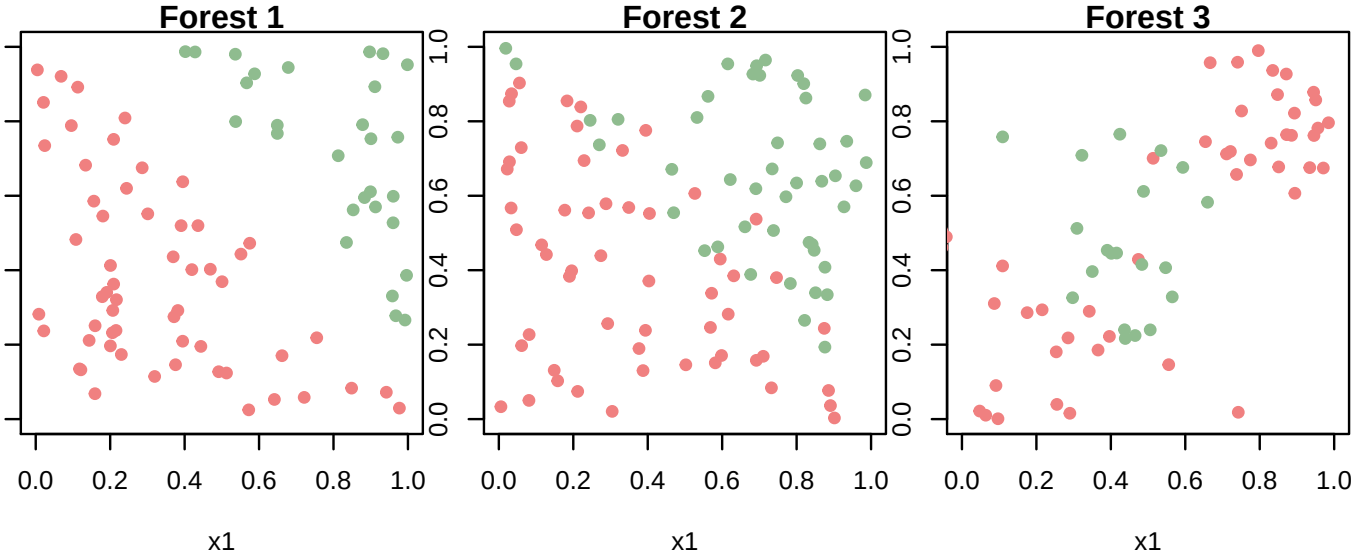
You will get to know

- Maximal margin classifier
- Support vector classifier
- Support vector machines
- Extensions
- Comparisons

and learn how to apply all that.

Motivating example

- 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 with the tree positions in three forests is visualized in the figures below. Orange: redwood tree; green: pine tree.



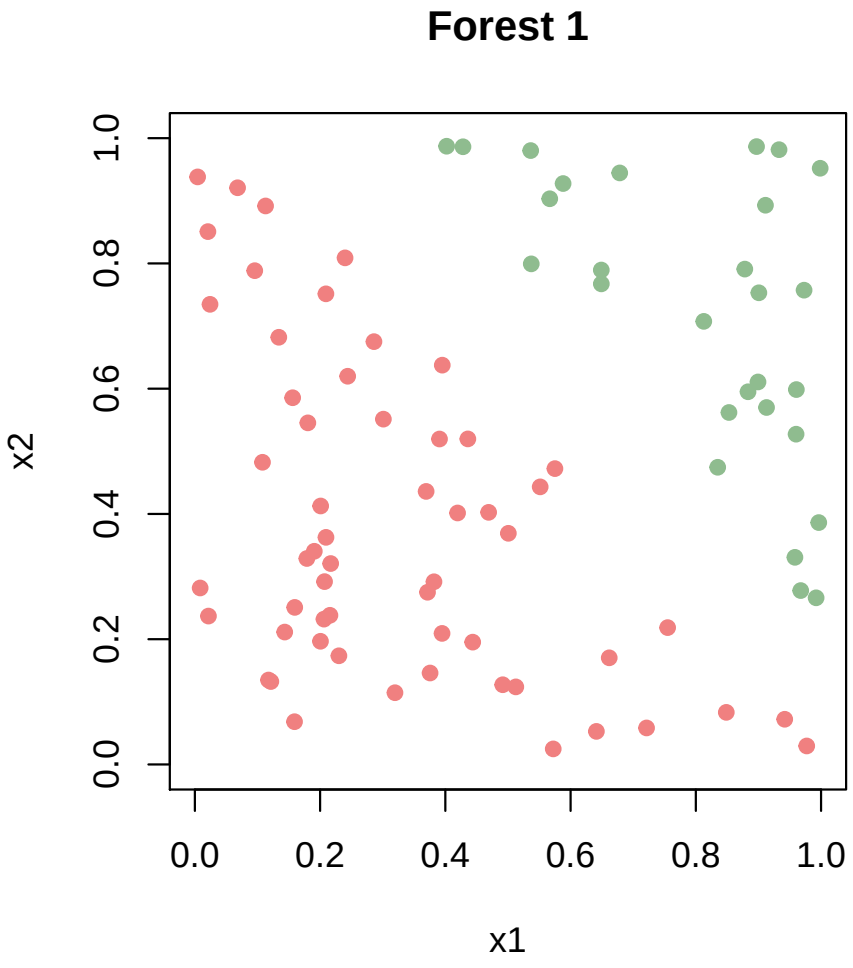
- You want to build one continuous fence to separate the two tree types in each of the three study areas. **Where should you build the fence?**

Three cases:

- Forest 1 seems easy: The orange and green points are clearly separated and the fence can be built anywhere inside the band that separates them. However, we can draw infinitely many straight lines that all separate the two tree types, and we should take into account that the trees reproduce and that we want future pines and future redwoods to grow up on the correct side of the fence.
- Forest 2 is a bit more complicated: A linear fence still seems like a good idea, but in this case the two tree types cannot be perfectly separated. You have to allow some of the trees to be on the wrong side of the fence.
- Forest 3 is the most complex: It is not possible to separate the two tree types by a straight line without getting a large number of misclassifications. Here, a circular fence around the pine trees seems like a reasonable choice.

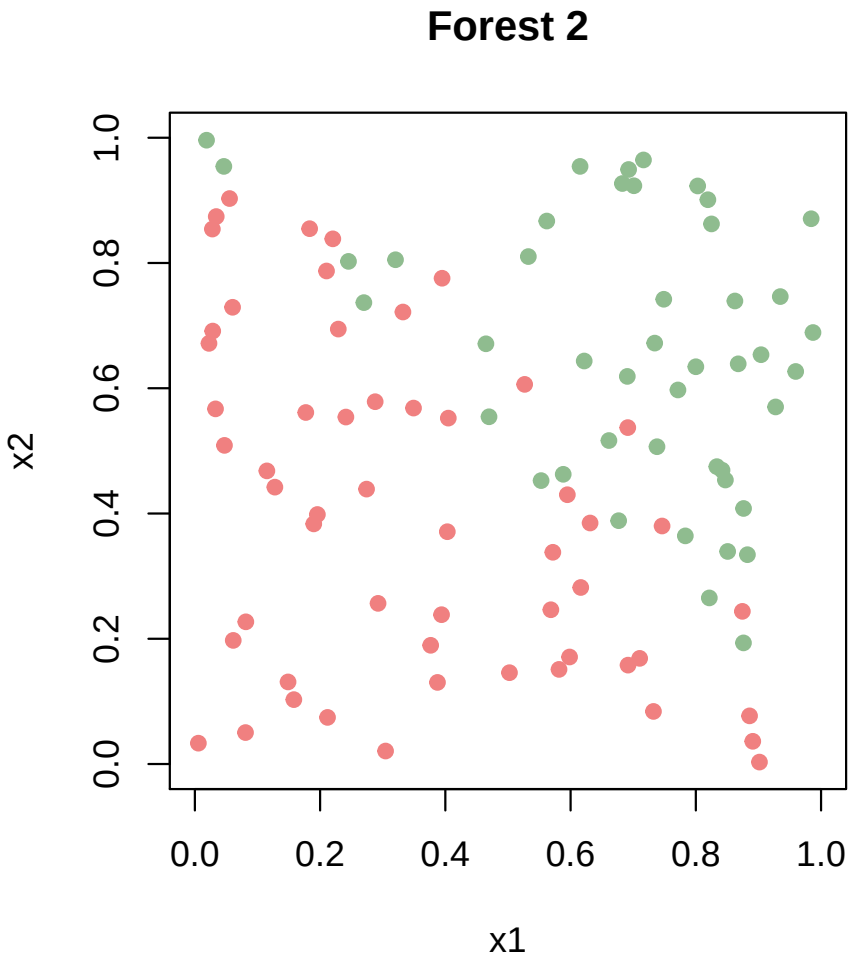
Forest 1 illustrates the problem of finding an **optimal separating hyperplane** for a dataset.

Topic: **Maximal Margin hyperplanes**



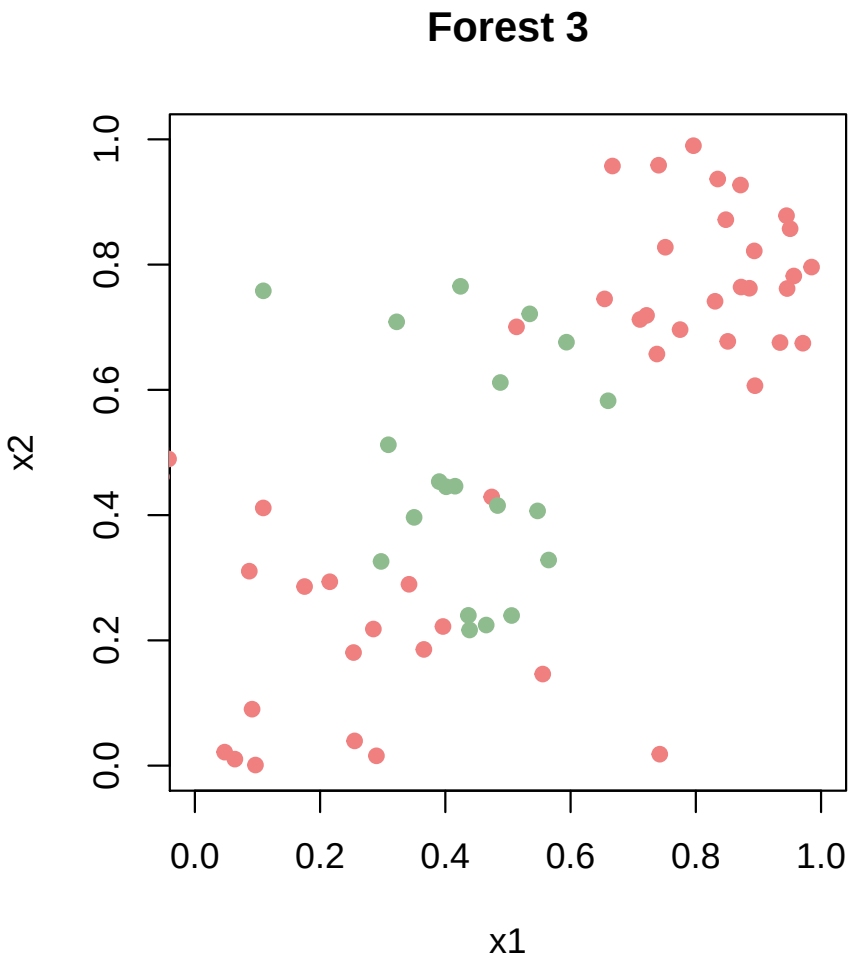
You are also going to learn how you can find an optimal separating hyperplane when your data cannot be perfectly separated by a straight line, as in Forest 2.

Topic: **Support Vector Classifier** or **Soft Margin Classifier**



The Support vector classifier can be generalised to an approach that produces non-linear decision boundaries. This is useful when the data is distributed as illustrated in Forest 3.

Topic: **Support Vector Machines** (SVMs)



Why are we interested in an optimal hyperplane or a separating curve?

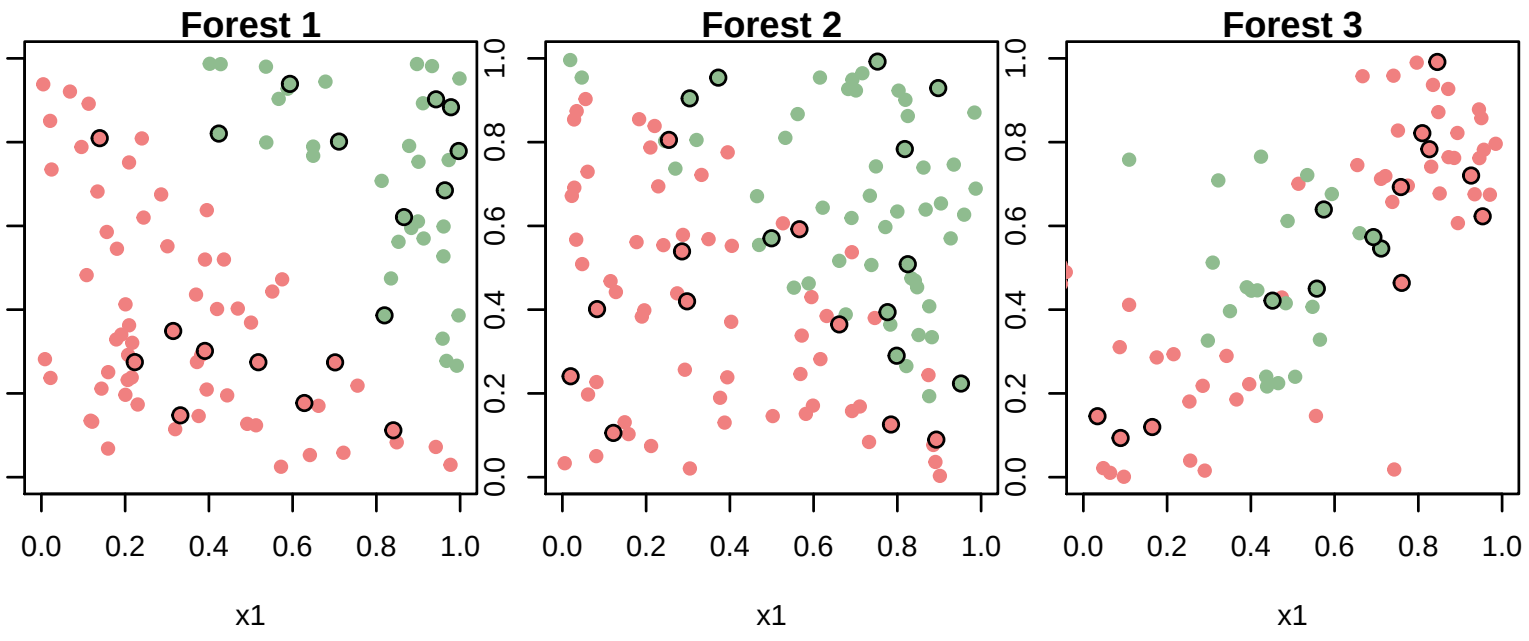
$\rightarrow$ We want to use it for classification.

- In our example: Is it likely that a random seed found at location $(0.8, 0.4)$ becomes a redwood or a pine tree given the observed data?
- Classification of a new observation based on which side of the decision boundary it falls into.

Note: In this module we look at **binary classification**, but extensions to more than two classes are briefly mentioned.

- We will use the three forest examples throughout this module.
- The two covaraites (x_1, x_2) are the coordinates of the trees.
- The response is either *pine* $(y = 1)$ or *redwood* $(y = -1)$.
- **Goal**: to make a classifier for random seeds that we find on the ground for each of the three forests. The locations of the seeds are shown in the figure below (black circles).

- The point patterns of the known locations can be thought of as the training set.
- The point pattern generated by the black circles can be thought of as the test set.



Maximal Margin Classifier

Hyperplane

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

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

and is a $p-1$ dimensional subspace of $\mathbb{R}^p$.

Recap:

- 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.
- The vector $\beta = (\beta_1, \dots, \beta_p)$ (not including β_0) is called the normal vector and points in the direction orthogonal to the hyperplane.

If a point x satisfies

- $\beta_0 + \beta^\top x > 0$ it lies on one side of the hyperplane
- $\beta_0 + \beta^\top x < 0$ it lies on the opposite side of the hyperplane.
- $\beta_0 + \beta^\top x = 0$ it lies on the hyperplane (by definition!).
- The signed distance d of any point x to the hyperplane is given by

$$d = \frac{1}{||\beta||}(\beta_0 + \beta^\top x) \ ,$$

  where $||\beta||^2 = \sum_{j=1}^{p} \beta_j^2 = 1$ is the (squared) length of β (Euclidian norm), see also here.
- See board for a graphical description.

Assumptions

- Assume that we have n training observations with p predictors

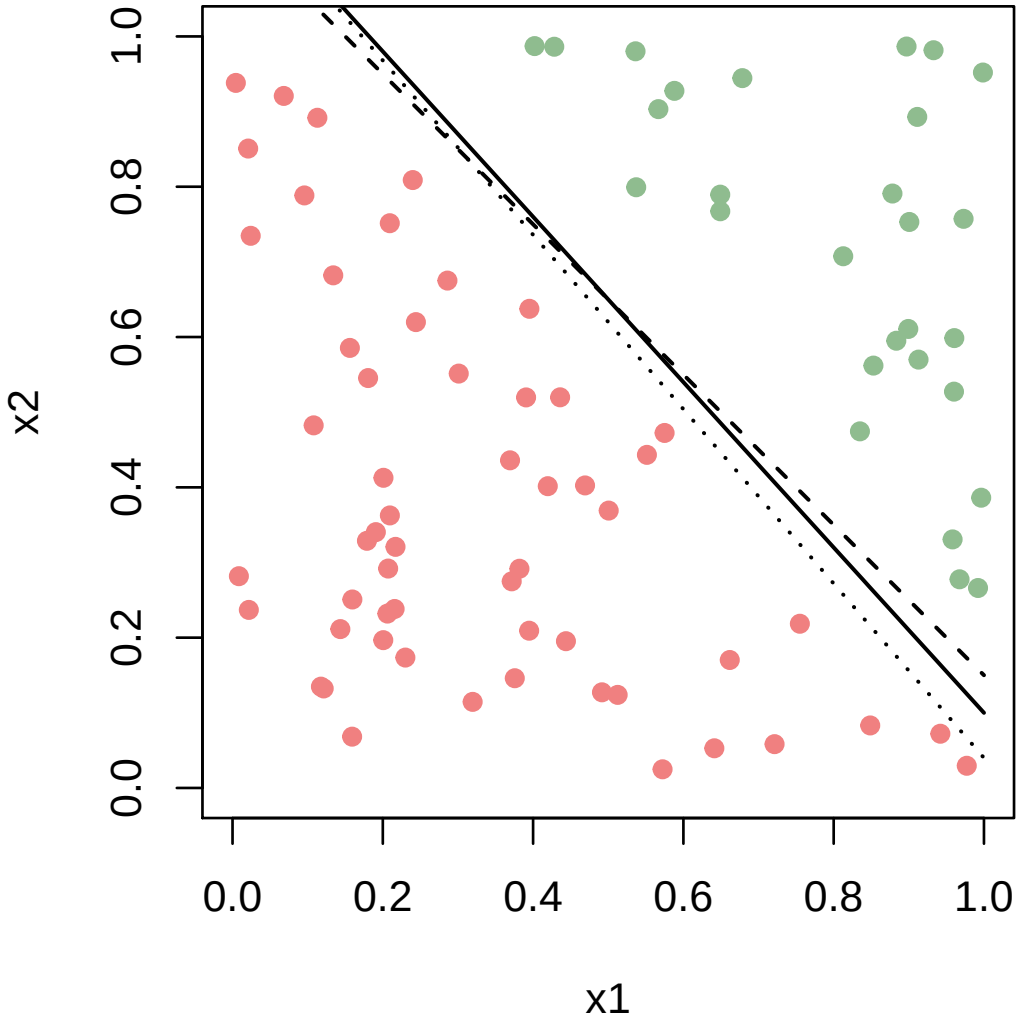
$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix} .$$

- The responses y fall into two classes $y_1, ..., y_n \in \{-1, 1\}$.

- It is possible to separate the training observations perfectly according to their class.

Possible hyperplanes (Forest 1)

Which is “best”?



Classification with a simple hyperplane

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

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} = \beta_0 + x_i^T \beta > 0$$

if $y_i = 1$ (green points) and

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} = \beta_0 + x_i^T \beta < 0$$

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

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

$$y_i(\beta_0 + x_i^T\beta) > 0 \ .$$

The hyperplane thus leads to a *natural classifier*, depending on the side of the hyperplane where the new observation lies.

Analogous for p predictors: The class y^* of a new observation $x^* = (x_1^*, ..., x_p^*)$ is assigned depending on the value $f(x^*) = \beta_0 + \beta_1 x_1^* + \beta_2 x_2^* + ... + \beta_p x_p^*$.

Hyperplane classifier:

$$y^* = \begin{cases} 1 \ , & \text{if } f(x^*) > 0 \ , \\ -1 \ , & \text{if } f(x^*) < 0 \ . \end{cases}$$

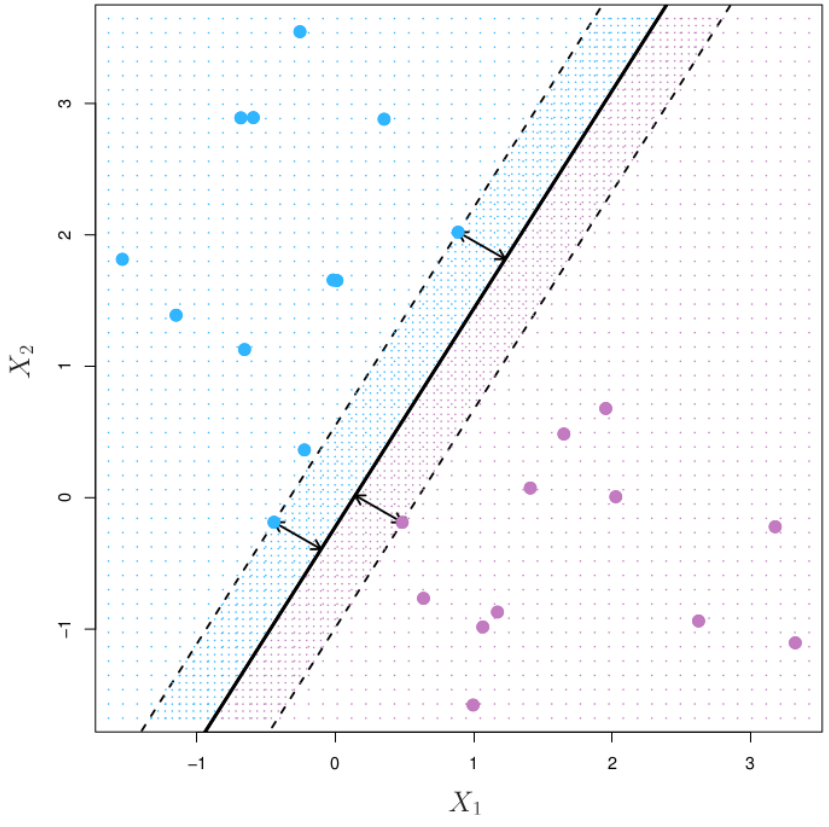
Which hyperplane is best?

- In the above figure we plotted three possible hyperplanes.
- In general, if data are linearly separable, infinitely many possible separating hyperplanes exist.
- Natural choice: the **maximal margin hyperplane**, which maximises the distance from the training observations.

Procedure:

- Compute 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 (the **margin**).
- We want to maximize this margin.

We have an **optimization problem** to maximize the width of the margin:



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**:

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

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

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

where M is the width of the margin.

Observe:

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

- In the ISLR Figure 9.3, the three equidistant point 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 Hastie, Tibshirani, and Friedman (2009) Section 4.5.2 (*Optimal Separating Hyperplanes*), 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[1].

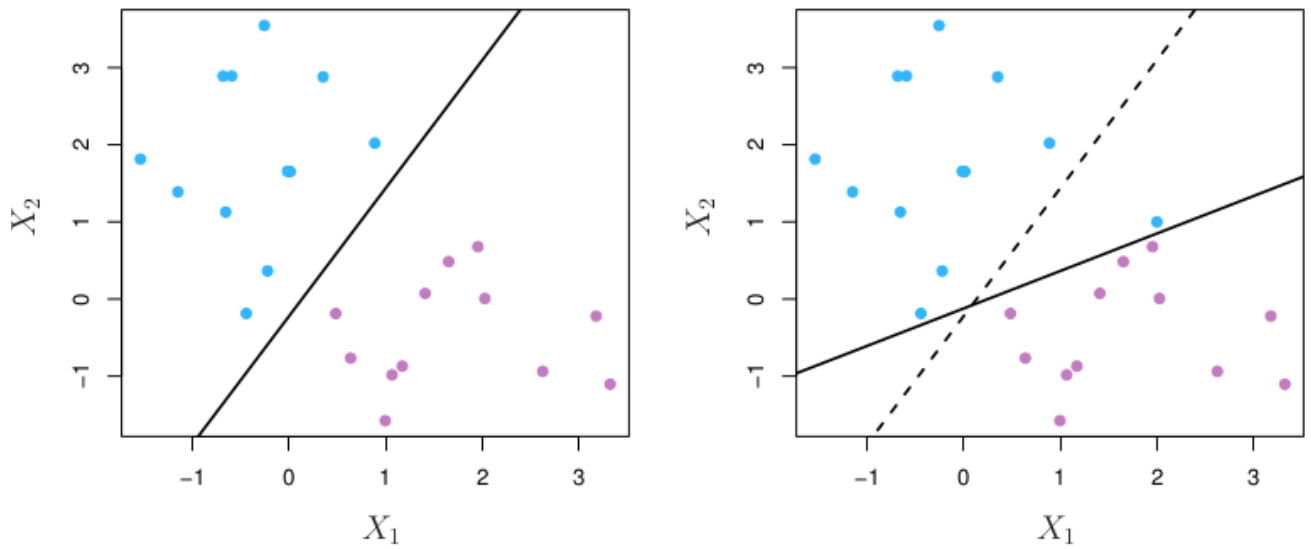
Q: But why do we need to solve the optimization problem, if the solution only depends on a few support vectors?

A: We do of course have to solve the optimization problem to identify the support vectors and the unknown parameters for the separating hyperplane.

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

Questions

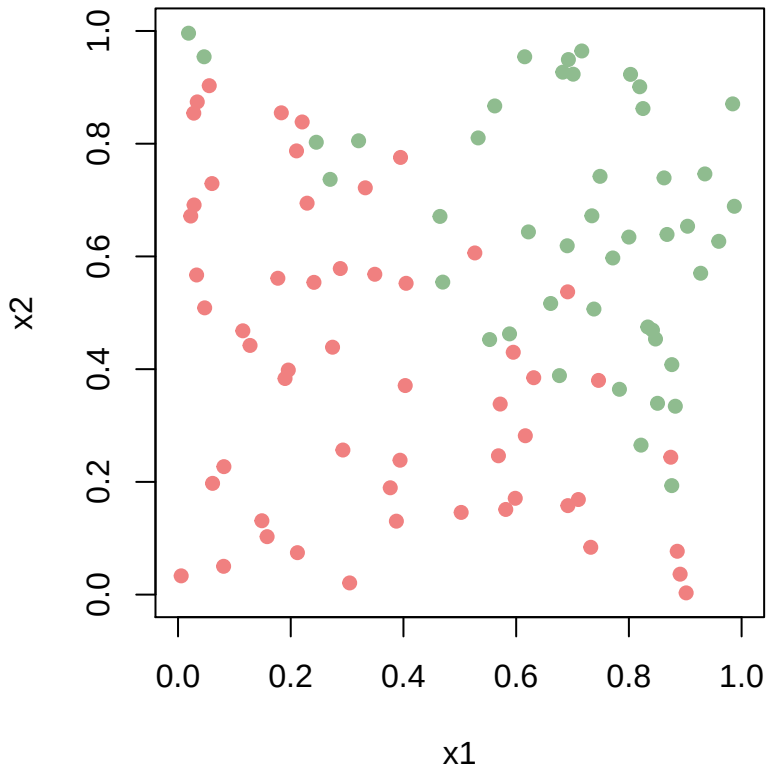
1. Explain briefly the idea behind the maximal margin classifier.
2. Is there any tuning parameters that needs to be chosen?
3. Look at the figure below. What could be the problem with the (maximal margin) hyperplane idea?



ISLR Figure 9.5

Support Vector Classifiers

For some data sets a separating hyperplane does not exist, the data set is *non-separable*. What then? Forest 2:



It is still possible to construct a hyperplane and use it for classification, but then we have to *allow some misclassification* in the training data.

- In some situations allowing for some misclassifications makes the class boundaries more robust to future observations (avoid overfitting).
- Even when the data are linearly separable[2], the separating hyperplane might not be the "best" hyperplane for us in terms of robustness.
- We relax the maximal margin classifier to allow for a *soft margin classifier* (=support vector classifier).

[2] in particular when $n \leq p + 1$, we can always find a separating hyperplane, unless there are exact feature ties across the class barrier, see Efron and Hastie, 2016.

Optimization problem

To obtain a **support vector classifier** we relax the conditions that we had for the maximal margin hyperplane by allowing for a "budget" C of misclassifications:

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

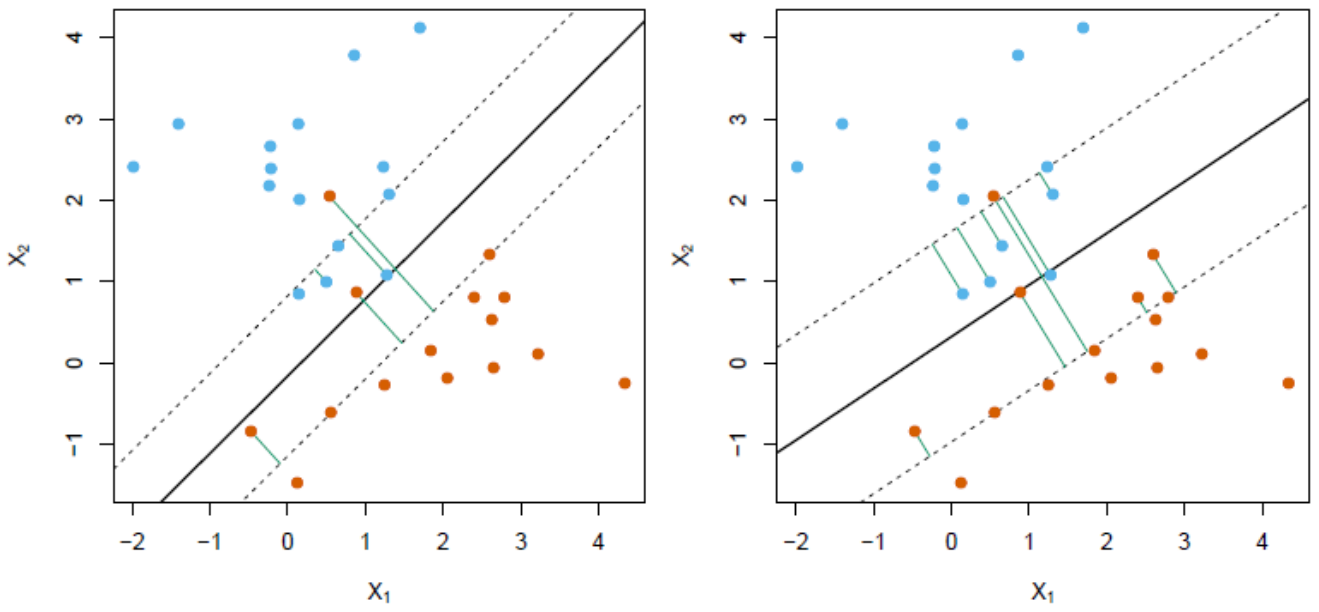
$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1,$$

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

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

- M is the width of the margin.
- $\epsilon_1, ..., \epsilon_n$ 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 side of the hyperplane. Less than C of the observations can be on the wrong side.

Figure 19.3 in Efron and Hastie (2016) gives a nice graphical representation of the soft margin classifier idea, where the violations (estimates of ϵ_i) are shown in green:



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

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

More on solving the optimization problem: Hastie, Tibshirani, and Friedman (2009) Section 12.2.1 (primal and dual Lagrange problem, quadratic convex problem).

- 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**. In fact, a noteworthy property of the solution is that

$$\hat{\beta} = \sum_{i \in \mathcal{S}} \hat{\alpha}_i x_i \ ,$$

  where $\mathcal{S}$ is a *support set*.

- The observations $x_i, i \in \mathcal{S}$ 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.

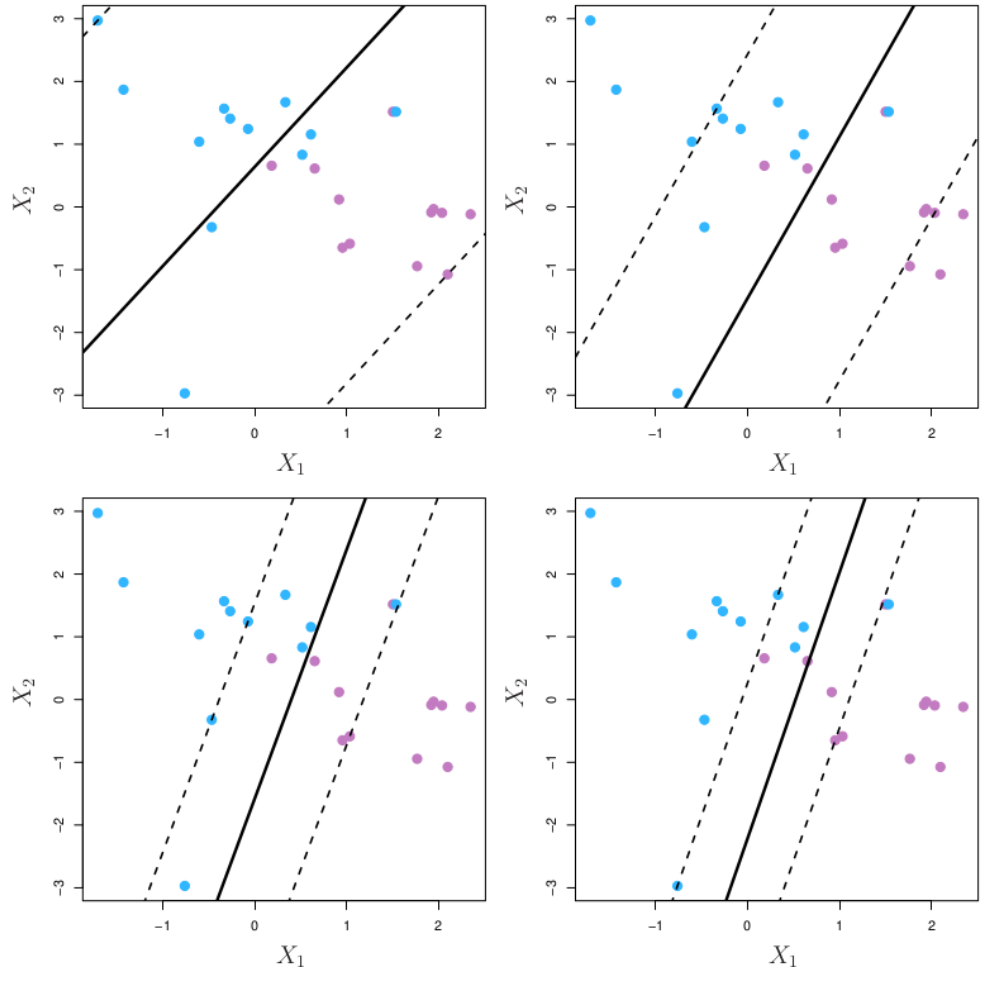
Questions

1. Should the variables be standardized before used with this method?
2. The support vector classifier only depends on the observations that violate the margin. How does C affect the width of the margin?
3. Discuss how the tuning parameter C affects the bias-variance trade-off of the method.

A:

Role of the tuning parameter C

ISLR Figure 9.7: From large C (top left) to small C (bottom right). As C decreases the tolerance for observations being on the wrong side of the margin decreases and the marin narrows.



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.

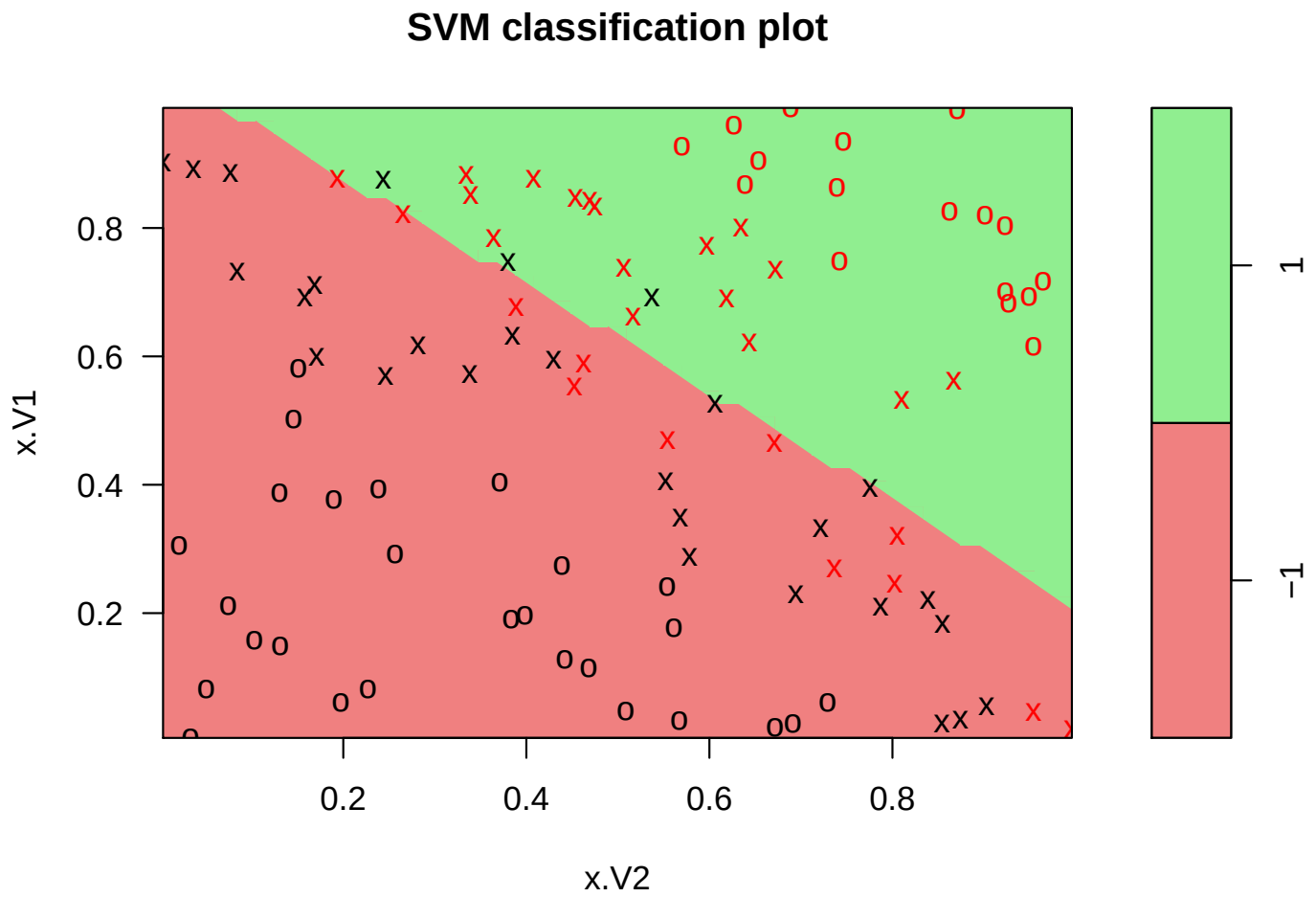
```
library(e1071)
forest2 = read.table(file = "forest2.txt")
seeds2 = read.table(file = "seeds2.txt")
train2 = data.frame(x = forest2[, 1:2], y = as.factor(forest2[, 3]))
test2 = data.frame(x = seeds2[, 1:2], y = as.factor(seeds2[, 3]))
```

The `svm` function uses a slightly different formulation than what we introduced above.

- We a *budget* of errors C, but in `svm` we instead have an argument `cost` that allows us to specify the cost of violating the margin. You can think of the cost as $\propto \frac{1}{C}$.
- 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 `scale=FALSE` (no scaling).

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



(Note: The decision boundary looks a bit strange, and x_1 is plotted on the (usual) y-axis and x_2 on the x-axis. Both problems are due to implementation, nothing for us to worry.)

```r
summary(svmfit_linear1)
```

```
##
## Call:
## svm(formula = y ~ ., data = train2, kernel = "linear", cost = 1,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  56
##
##  ( 28 28 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Observations

- The crosses in the plot indicate the support vectors, whose index can be obtained from

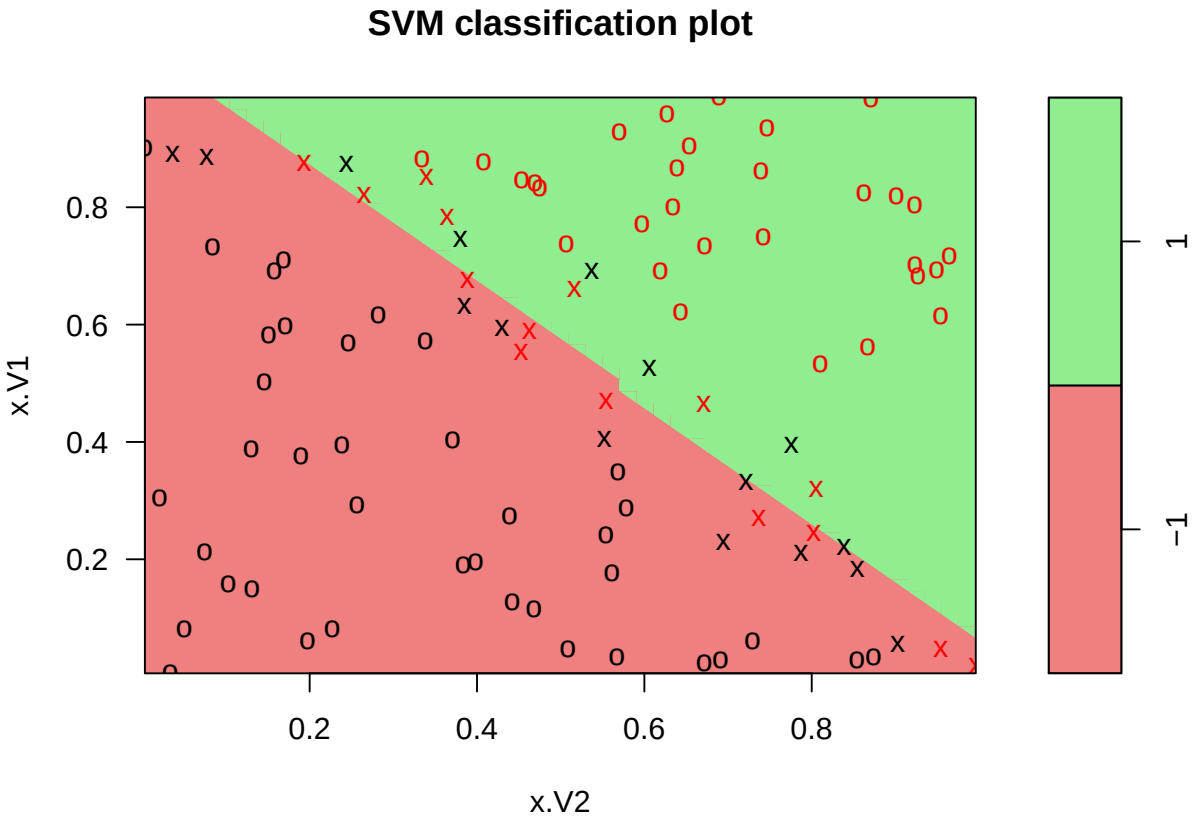
```
svmfit_linear1$index  #support vectors id in data set
```

```
##  [1]  1  2  4  6  9 10 16 21 26 27 28 40 44 53 55 57 58 65 67 72 76 77 80
## [24] 81 87 91 92 98  5  8 11 13 18 19 20 23 24 25 34 36 39 41 42 47 48 59
## [47] 61 62 70 71 75 78 88 93 95 96
```

- With `cost=1`, we have 56 support vectors, 28 in each class.
- All other data points are shown as circles.
- However, no explicit output for the linear decision boundary, and no margin width is given by `svm()`. Want to see how to find this? See the recommended exercises.

Next, we set `cost=100`:

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



```r
summary(svmfit_linear2)
```

```
##
## Call:
## svm(formula = y ~ ., data = train2, kernel = "linear", cost = 100,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  100
##
## Number of Support Vectors:  31
##
##  ( 15 16 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

Thus with `cost=100` we have 31 support vectors, i.e the width of the margin is decreased (remember: higher cost = lower budget to violate the boundaries).

Cross-validation to find an optimal `cost`

The `cost` is a tuning parameter. By using the `tune()` function we can perform 10-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(0.001,
    0.01, 0.1, 1, 5, 10, 50)))
summary(CV_linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.14
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.45 0.10801234
## 2 1e-02  0.23 0.12516656
## 3 1e-01  0.16 0.11737878
## 4 1e+00  0.15 0.10801234
## 5 5e+00  0.14 0.10749677
## 6 1e+01  0.15 0.09718253
## 7 5e+01  0.15 0.09718253
```

According to the `tune()` function we should set the cost parameter to 5. The function also stores the best model obtained and we can access it as follows:

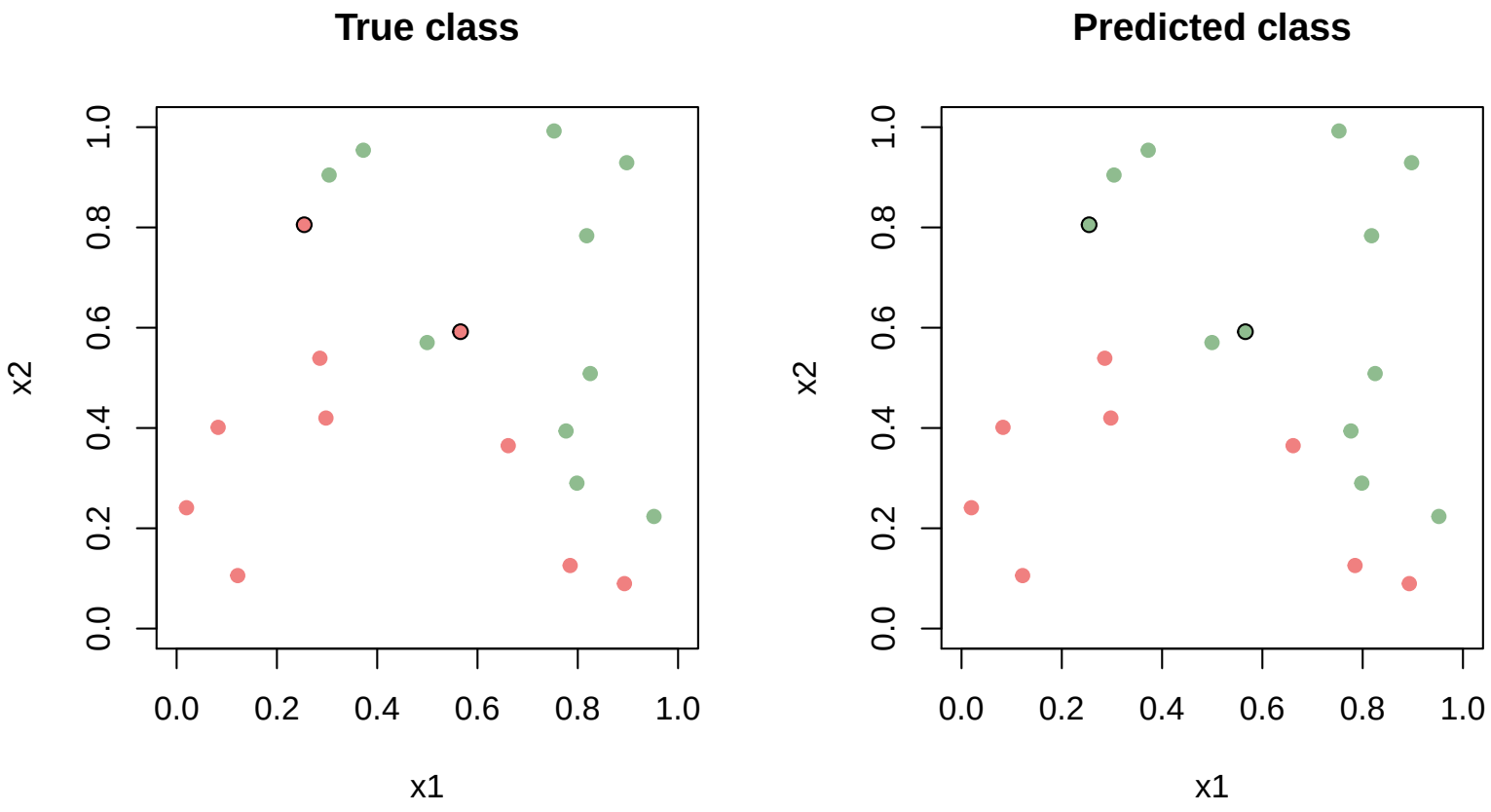
```
bestmod_linear = CV_linear$best.model
```

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  8  0
##      1   2 10
```

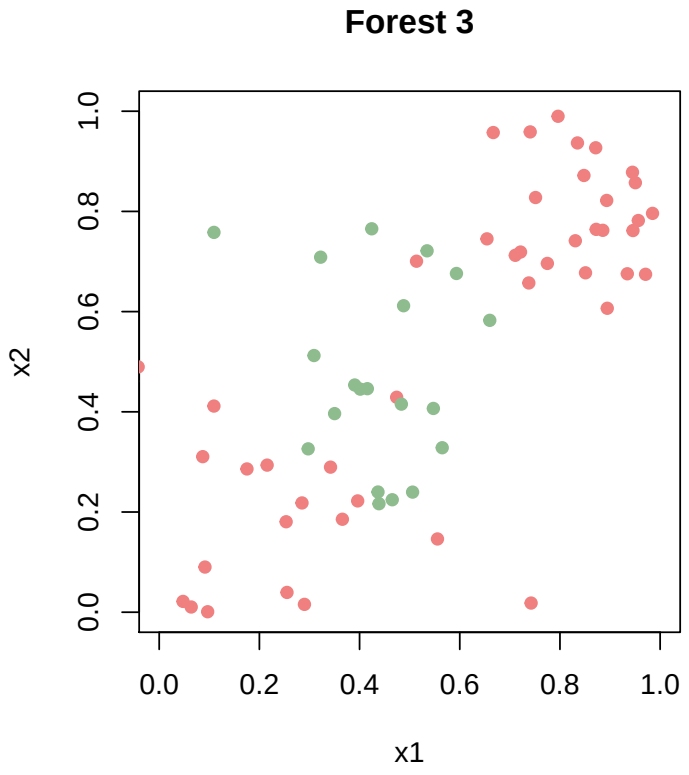
Thus two of the seeds are misclassified, the other 18 are ok.



The two misclassified observations are marked with a black circle. Unsurprisingly, they lie on the border between the green and the orange points. Why?

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.



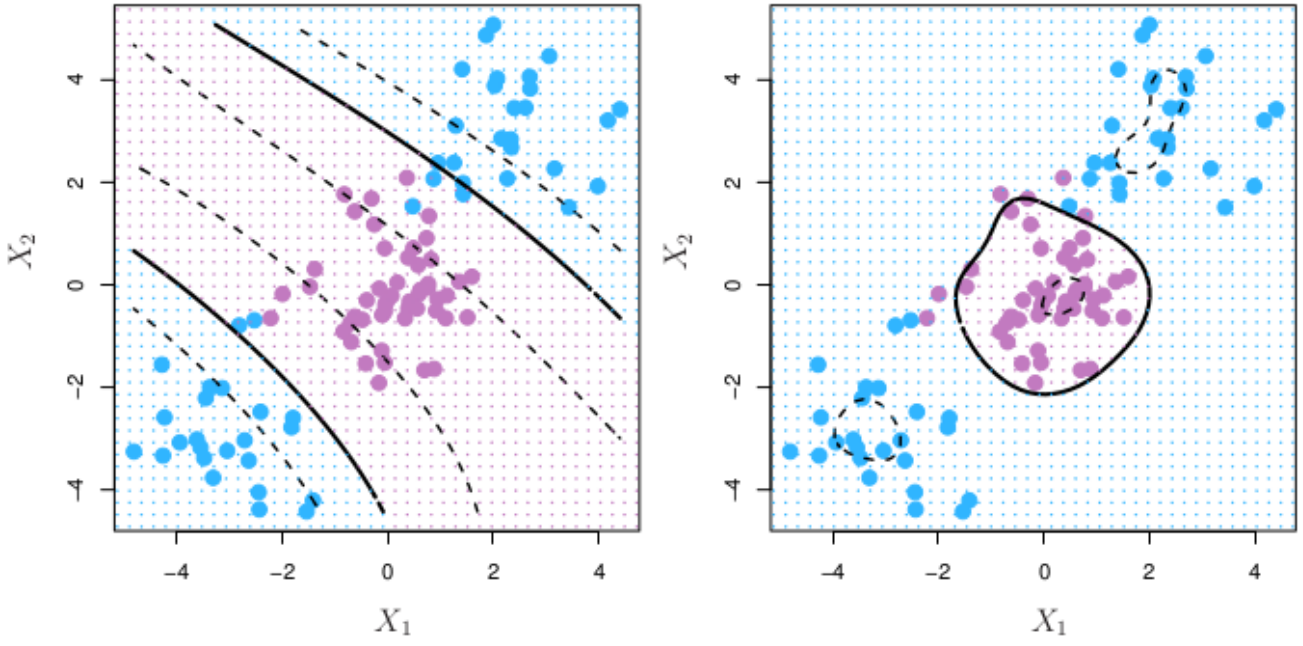
Expanding the feature space

- Recall from Module 7: We could fit non-linear regression curves by using a polynomial basis. This was a **linear regression in the transformed variables**, but non-linear in the original variables.
- **Idea**: Find a linear boundary in that high-dimensional space using
  - higher-order terms X_i^k
  - interaction terms $X_i X_{i'}$
  - other functions of X_i.
- This leads to a non-linear boundary in the original space.
- **Example**:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \beta_4 X_1^2 + \beta_5 X_2^2 = 0 \ .$$

Example

(ISRL Figure 9.9)



- **Left**: expanding feature space to include cubic polynomials ($x_1, x_2, x_1x_2, x_1^2, x_2^2, x_1^2x_2, x_1x_2^2x_1^3, x_2^3$, 9 parameters to estimate in addition to intercept), and also observe the margins (interpretation?).
- **Right**: radial basis function kernel - wait a bit.

Problems and better ideas

- Computation using polynomials quickly becomes unmanageable.
- More elegant idea is the use of *kernels*.
- But first we have to understand the role of *inner products* in Support Vector Classifiers.

Inner products

The *inner product* between two observations i and i' is defined as

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

The inner product encodes for the *similarity* between observations.

- Remember the the optimisation problem of finding the support vector classifier hyperplane. We have *not* explained how to solve the problem because this is outside the scope of this course.

- But we said that

$$\hat{\beta} = \sum_{i \in \mathcal{S}} \hat{\alpha}_i x_i$$

  for a support set $\mathcal{S}$.

- Thus the solution function $\hat{f}$ to the support vector classifier problem at a new observation x, can be expressed as

$$\begin{aligned} \hat{f}(x) &= \hat{\beta}_0 + x^\top \hat{\beta} \\ &= \hat{\beta}_0 + \sum_{i \in \mathcal{S}} \hat{\alpha}_i \langle x, x_i \rangle \ , \end{aligned}$$

  where α_i is some parameter and $i = 1, ..., n$.

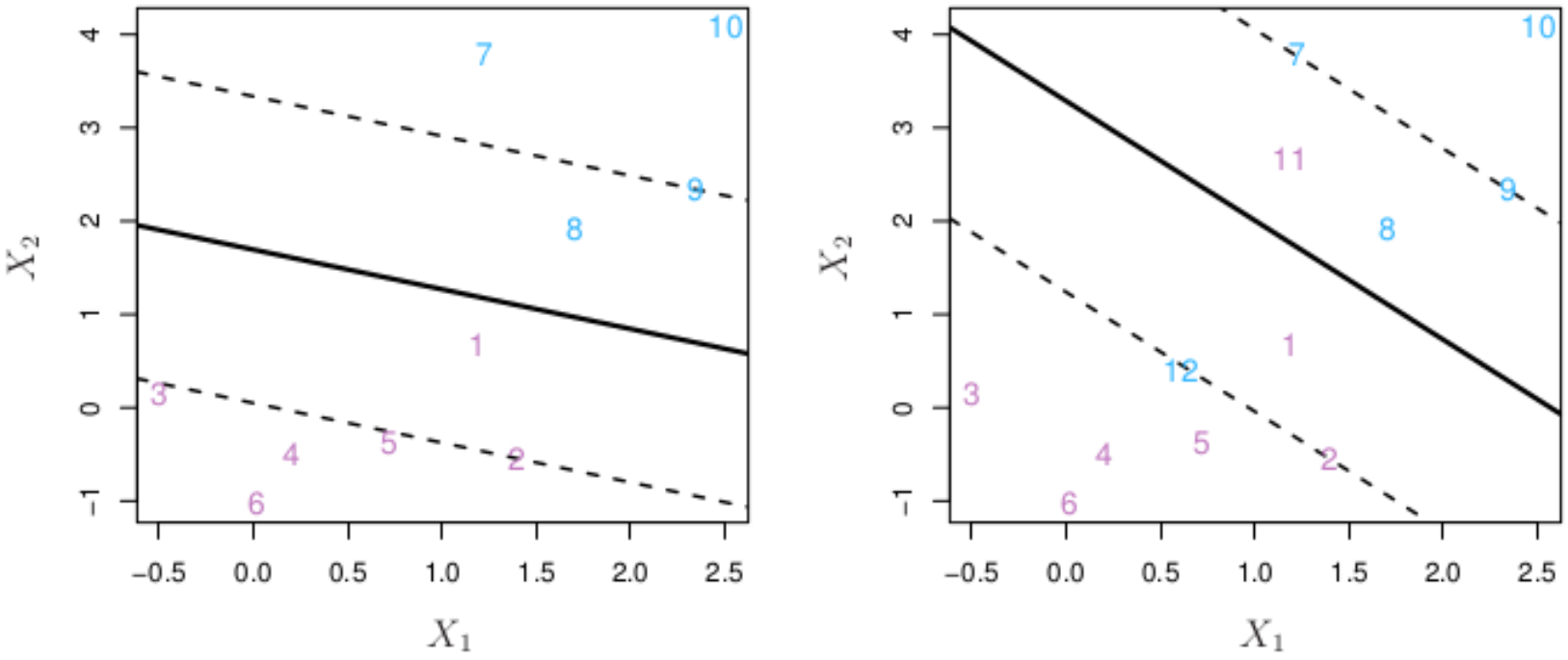
- This implies that to estimate the parameters $\beta_0, \alpha_1, ..., \alpha_n$ we only need to know the $\binom{n}{2}$ inner products $\langle x_i, x_i' \rangle$ between all pair of training observations (and their correct class)[3].
- Even better, $\alpha_i \neq 0$ for the support vectors x_i ($i \in \mathcal{S}$), while $\alpha_i = 0$ for all the rest.

This will bring us to another idea:

- All we need is to know to classify a new observations is the **similarity** to all the training observations, where similarity could be defined in any way.

[3]For the interested reader: See Section 12.2.1 in James et al (2013), or Eq. 19.22 and 19.23 of Efron and Hastie (2016).

Q: Find the support vectors



- **Q**: So why don't we just directly only use the support vectors?
- **A**:

Kernels

- Idea: replace the innerproduct $\langle x_i, x_{i'} \rangle$ as measure of similarity by a more general concept: Replace $\langle x_i, x_{i'} \rangle$ by a *kernel* $K(x_i, x_{i'})$, such that

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

- For the familiar linear case, the kernel is simply the inner product (*linear kernel*) $K(x_i, x_i') = \sum_{j=1}^{p} x_{ij} x_{i'j}$.
- If we want a more flexible decision boundary we could instead use a *polynomial kernel* of degree $d > 1$:

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

which computes the inner products needed for d-dimensional polynomials. Try it for $p = 2$ and $d = 2$ (see exercises).

- By using *non-linear kernels*, the resulting classifier is a *support vector machine*.
- The nice thing here is that we only need to calculate the kernels, *not the basis functions.*
- We only need to compute $K(x_i, x_{i'})$ for the $\binom{n}{2}$ distinct pairs – without explicitly working in an enlarged feature space. This is very useful when there are *many features*, i.e., $p \geq n$.

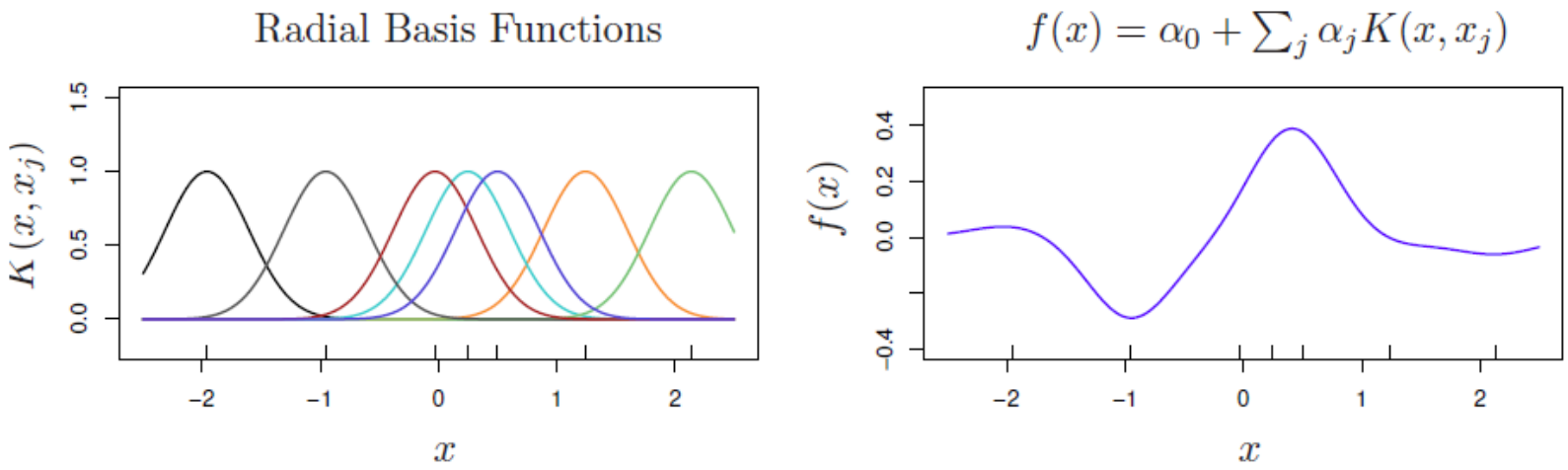
The radial kernel

- 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).
- Interestingly, the radial kernel 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" (but we have the parameter γ and the budget parameter C that we have to decide on).
- Connection to a multivariate normal density, where $\gamma \propto 1/\sigma^2$ (σ^2 variance in normal distribution). If γ is small (similar to large variance in the normal distribution) the decision boundaries are smoother than for larger γ.

- The radial kernel is convenient if we want a circular decision boundary. See Figure 19.5 in Efron and Hastie (2016):



- γ and our budget can be chosen by cross-validation.
- Remark: the mathematics behind this is based on *reproducing-kernel Hilbert spaces* (see page 384 of Efron and Hastie (2016) for a glimpse of the theory).

Study Figures 19.5 and 19.6 (page 383) in Efron and Hastie (2016) to see how the radial kernel can make smooth functions.

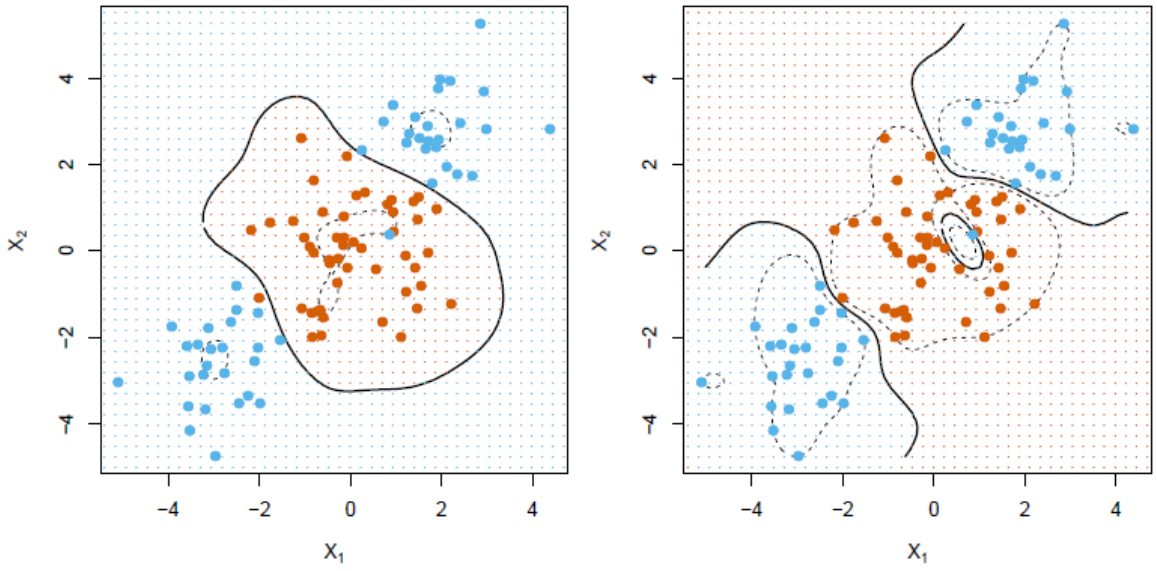


Figure 19.6 Simulated data in two classes in $\mathbb{R}^2$, with SVM classifiers computed using the radial kernel (19.11). The left panel uses a larger value of B than the right. The solid lines are the decision boundaries in the original space (linear boundaries in the expanded feature space). The dashed lines are the projected margins in both cases.

If you want to download the whole book you can do this here:

Computer Age Statistical Inference

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)

$$\text{maximize}_{\beta_0,\alpha_1,...,\alpha_n,\epsilon_1,...,\epsilon_n,} \quad M$$

$$y_i \cdot f(x_i) \geq M(1-\epsilon_i) \quad \forall i = 1, ..., n \ ,$$

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

where

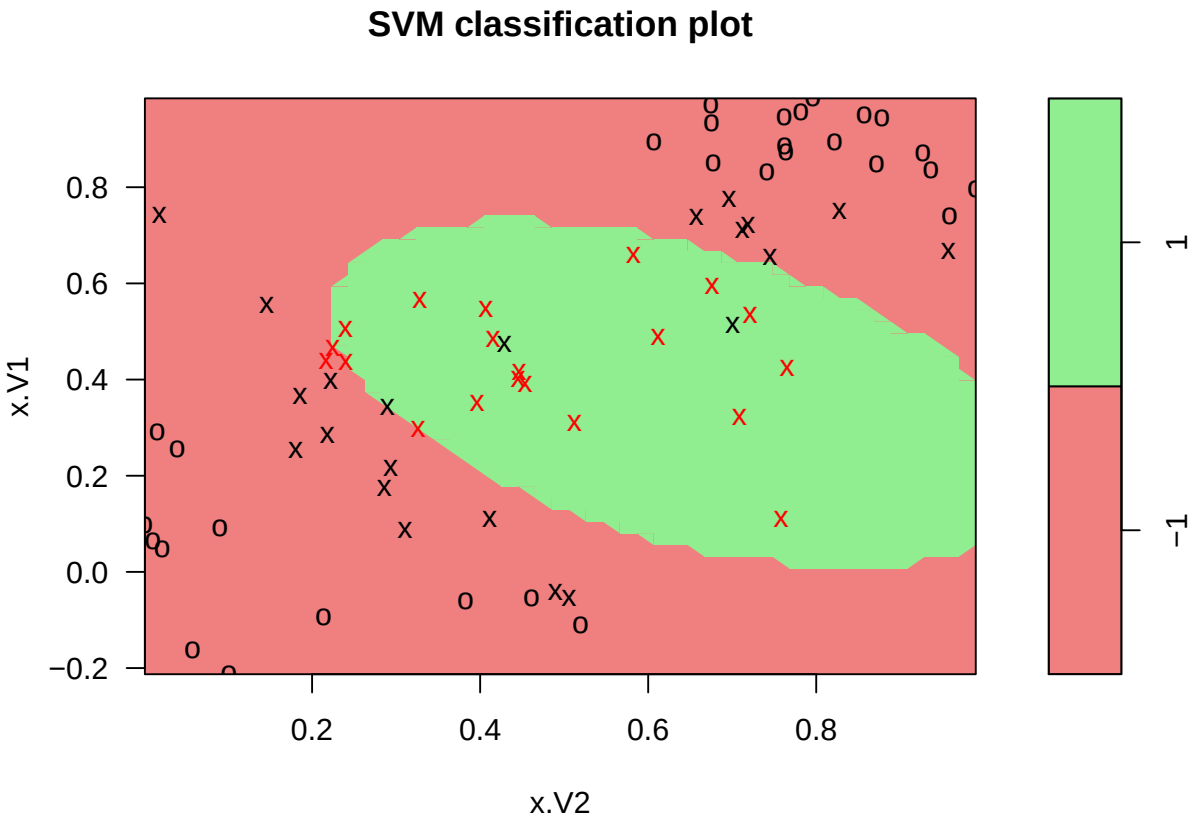
$$f(x_i) = \beta_0 + \sum_{l=1}^{n} \alpha_l K(x_i, x_l) \ .$$

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", cost = 10,
    scale = FALSE)
plot(svmfit_kernel1, train3, col = c("lightcoral", "lightgreen"))
```

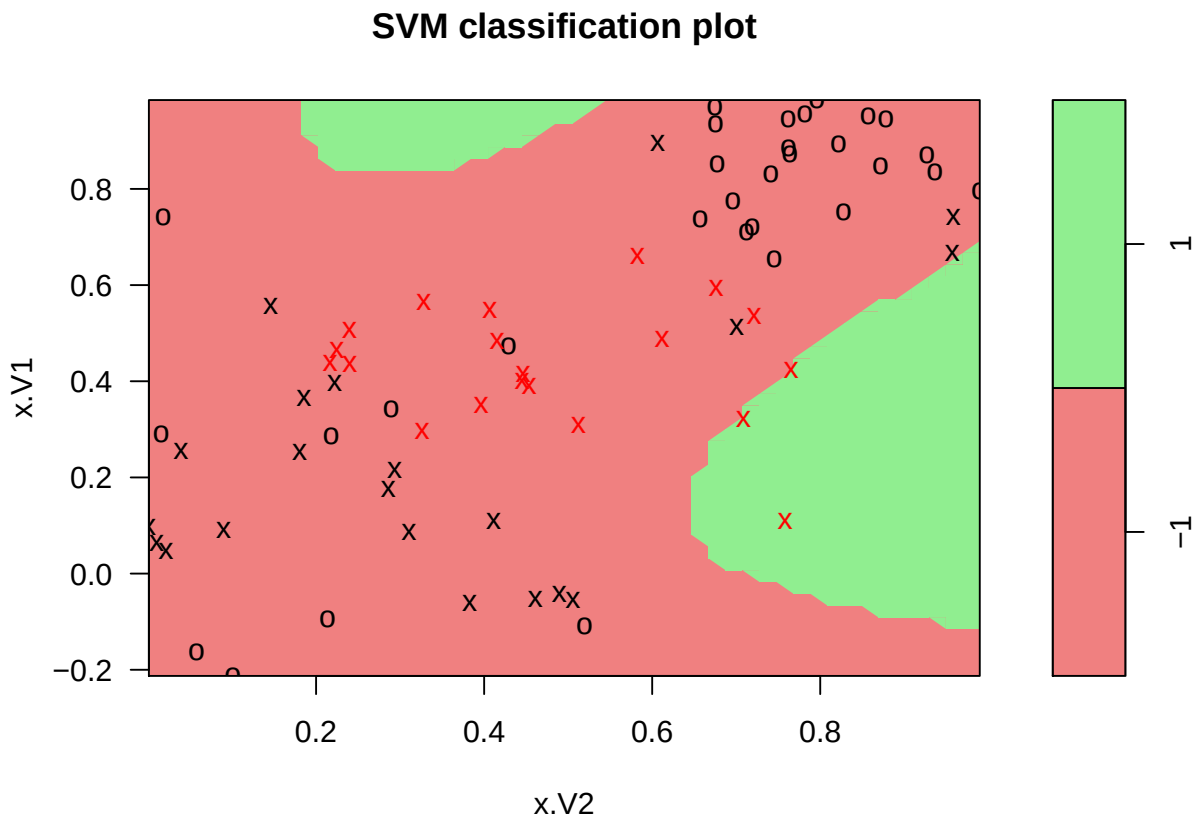


```r
summary(svmfit_kernel1)
```

```
##
## Call:
## svm(formula = y ~ ., data = train3, kernel = "radial", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  42
##
##  ( 22 20 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

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

```
svmfit_kernel2 = svm(y ~ ., data = train3, kernel = "polynomial", degree = 4,
    cost = 10000, scale = FALSE)
plot(svmfit_kernel2, train3, col = c("lightcoral", "lightgreen"))
```



```r
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
##      coef.0:  0
##
## Number of Support Vectors:  40
##
##  ( 21 19 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

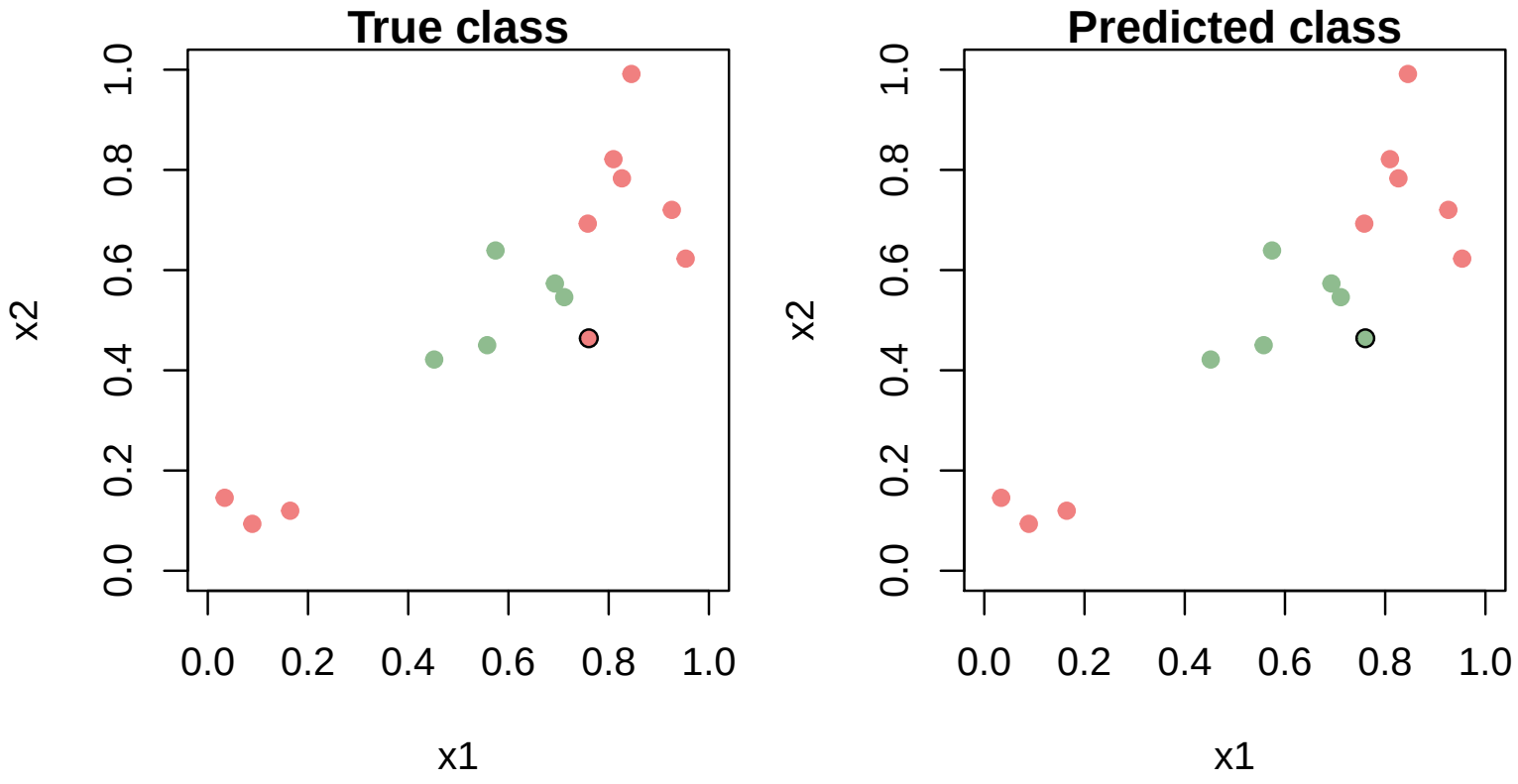
For this dataset a radial kernel is a natural choice: A circular decision boundary seems like a good idea. Thus, we proceed with `kernel='radial'`, and use the `tune()` function to find the optimal tuning parameter C:

```
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, 1000)))
summary(CV_kernel)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.1089286
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.2732143  0.1472658
## 2 1e-02 0.2732143  0.1472658
## 3 1e-01 0.2732143  0.1472658
## 4 1e+00 0.1500000  0.1315463
## 5 5e+00 0.1357143  0.1226248
## 6 1e+01 0.1214286  0.1298110
## 7 1e+02 0.1089286  0.1066291
## 8 1e+03 0.1767857  0.1226970
```

The optimal C is 100. Next, we predict the class label of the seeds in the test set with a model with $C = 100$, make a confusion table and plot the results:

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



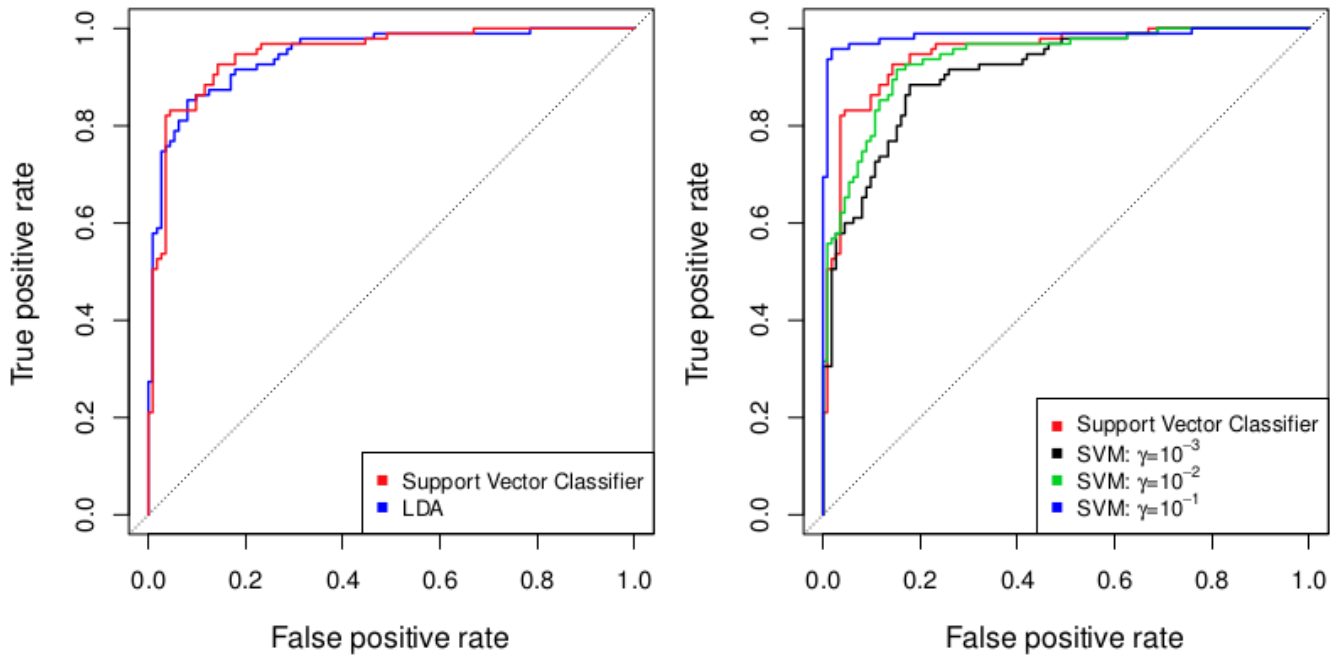
```
table(predict = ypred_kernel, truth = test3[, 3])
```

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

Only one seed is misclassified.

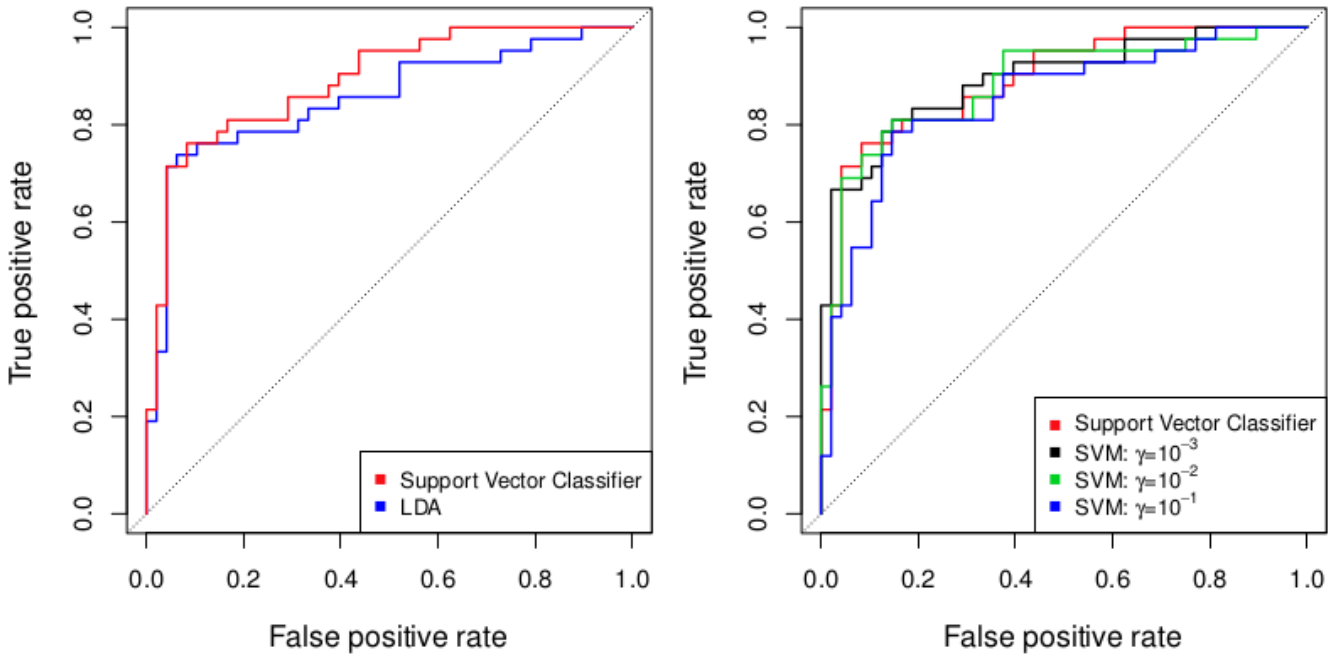
Tuning parameter example

Heart data - predict heart disease from $p = 13$ predictors.
Training errors as ROC and AUC.



Q: How are the ROC curves formed?

Heart data - test error.



Extensions

More than two classes

What if we have k classes? Two popular approaches:

- OVA: *one-versus-all*. Fit k different two-class SVMs $f_k(x)$ where, each time, one class is compared to the ensemble of all other classes. Classify a test observation to the class where $f_k(x^*)$ is largest.
- OVO: *one-versus-one*. `libsvm` uses this 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 for a given x^* is chosen).

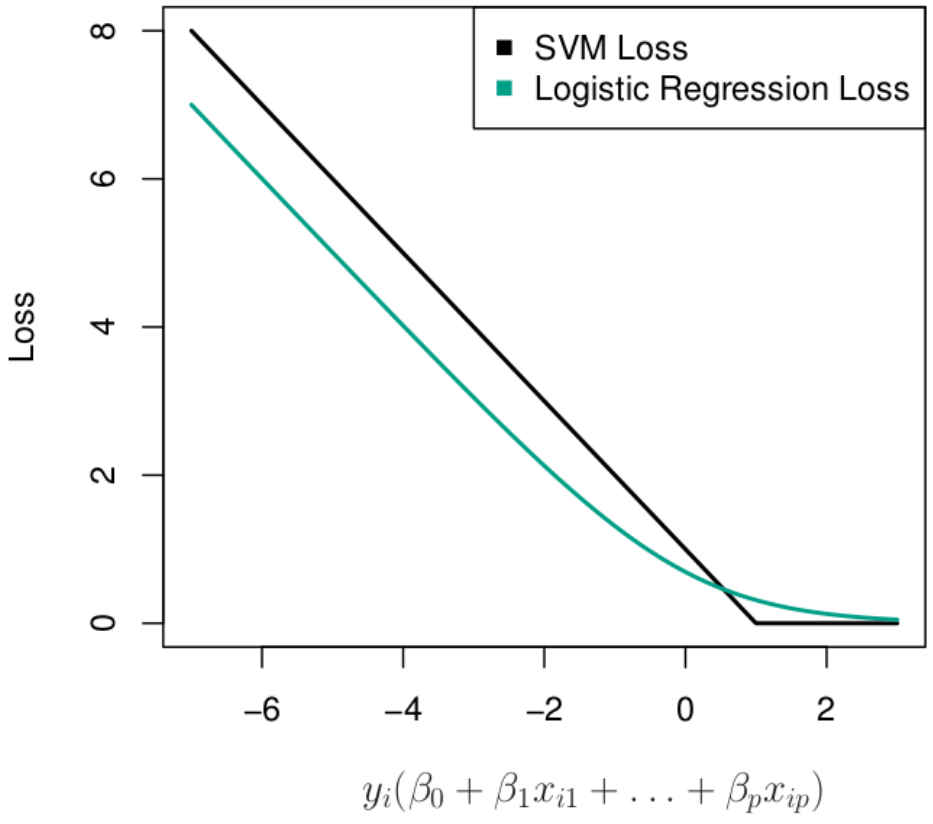
Comparisons: SVM and logistic regression

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

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

- The margin now corresponds to $M = 1$, and its width is determined by $\sum_{j=1}^{p} \beta_j^2$.
- 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 small βs and more violations=high bias, but low variance.
- Small λ gives large βs and less violations=low bias, but high variance.

Interestingly, the loss functions for a support-vector classifier with $f(X) = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p$ and for logistic regression using the same set of predictors *look very similar*:



Hinge loss:

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

- For comparison a logistic regression loss function would be (binomial deviance with $-1, 1$ coding of y)

$$\log(1 + \exp(-y_i f(x_i))) \ .$$

- 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
- Of course, it is possible to extend the logistic regression to include non-linear terms, and *also a ridge penalty*.

When to use SVM?

- If classes are nearly separable SVM will perform better than logistic regression. (Also LDA will perform better than logistic regression; what is the problem with logistic regression?)
- Otherwise, a ridge penalty version of logistic regression is **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* are also possible, but more computationally expensive (and traditionally less used).

Summing up

- We use methods from computer science, not probability models, but it also 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 (remember ridge-type penalty does not shrink to zero).
- The budget must be chosen wisely, and a bad choice can lead to overfitting.
- Not so easy to get class probabilites from SVM (what is done is actually to fit a logistic regression after fitting SVM).

Further reading

- Videoes on YouTube by the authors of ISL, Chapter 9, and corresponding slides
- Solutions to exercises in the book, chapter 9
- Chapters 12.1-12.3 in Hastie, Tibshirani, and Friedman (2009).
- Chapter 19 (Support-Vector Machines and Kernel Methods) in Efron and Hastie (2016).

References

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

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning*. 2nd ed. Vol. 1. Springer series in statistics New York.

James, G., D. Witten, T. Hastie, and R. Tibshirani. 2013. *An Introduction to Statistical Learning with Applications in R*. New York: Springer.