TMA4268 Statistical Learning V2019

Module 9: SUPPORT VECTOR MACHINES

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

week 11 2019

Contents

Last changes: (13.05 typos from student, 09.03: first version)

#Introduction

This field dates back to the 1990s in computer science, and in this presentation we put emphasis on the underlying motivation and connections to linear algebra, optimization theory and statistics.

We will only cover classification, and in particular two-class problems.

Learning material for this module

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

- Classnotes 11.03.2019

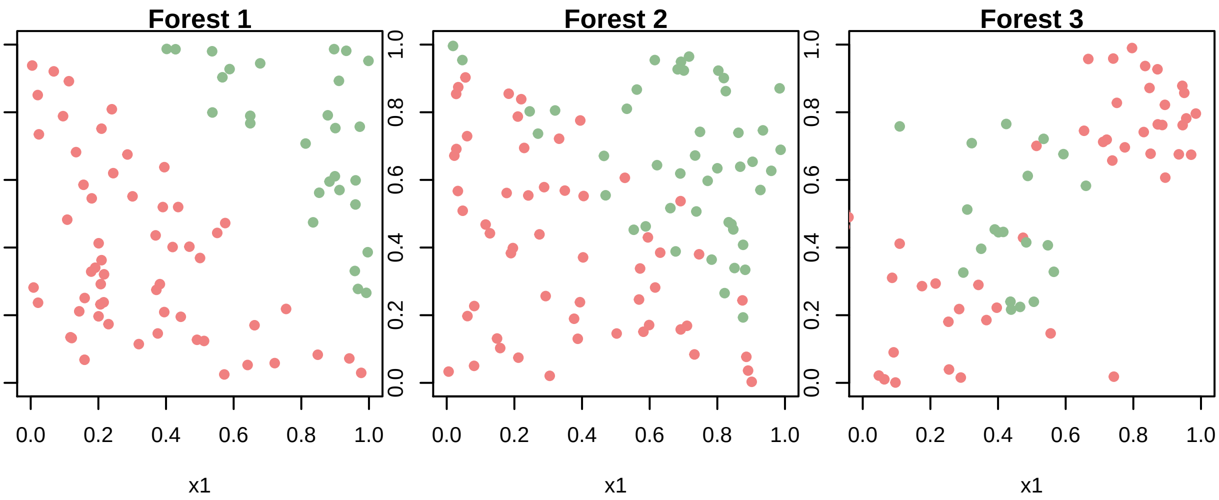
Some of the figures in this presentation are taken from (or are inspired by) "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

Topics

- Motivation
- Maximal margin classifier
- Support vector classifier
- Support vector machines
- Extensions
- Comparisons
- Summing up
- Recommended exercises
- R Packages
- References

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.

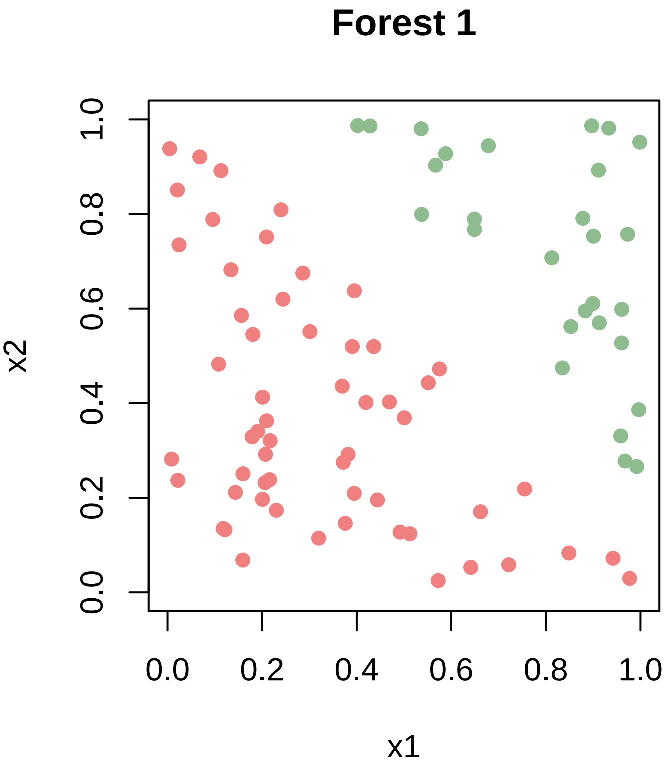


Assume you want to build one continuous fence to separate the two tree types in each of the three study areas. **Where should you build the fence?**

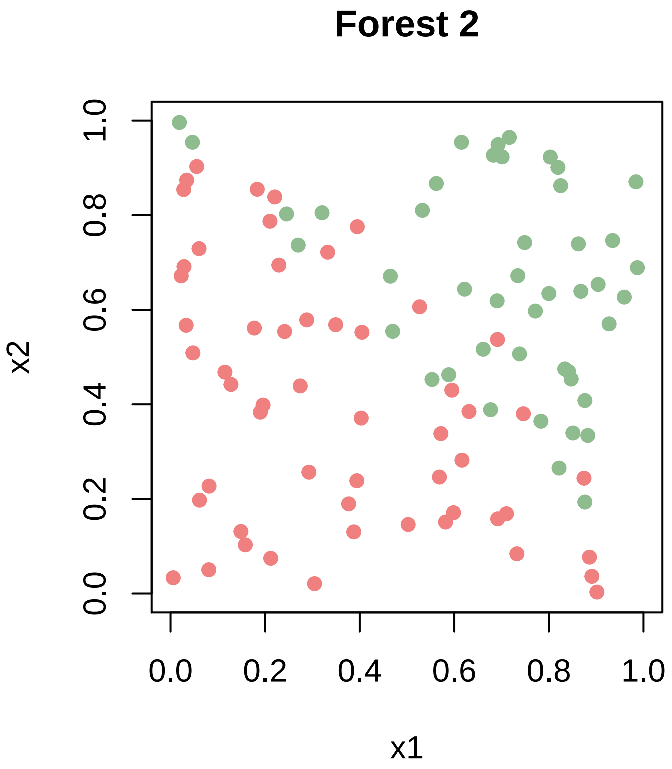
For Forest 1 the choice seems easy: The orange and green points are clearly separated and the fence can be built anywhere inside the band that separates them. However, we can draw infinitely many straight lines that all separate the two tree types, and we should take into account that the trees reproduce and that we want future pines and future redwoods to grow up on the correct side of the fence.

The problem gets more complicated for Forest 2. A linear fence still seems like a good idea, but in this case the two tree types can not be perfectly separated. You have to allow some of the trees to be on the wrong side of the fence. The complexity of the problem is further increased for Forest 3. 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.

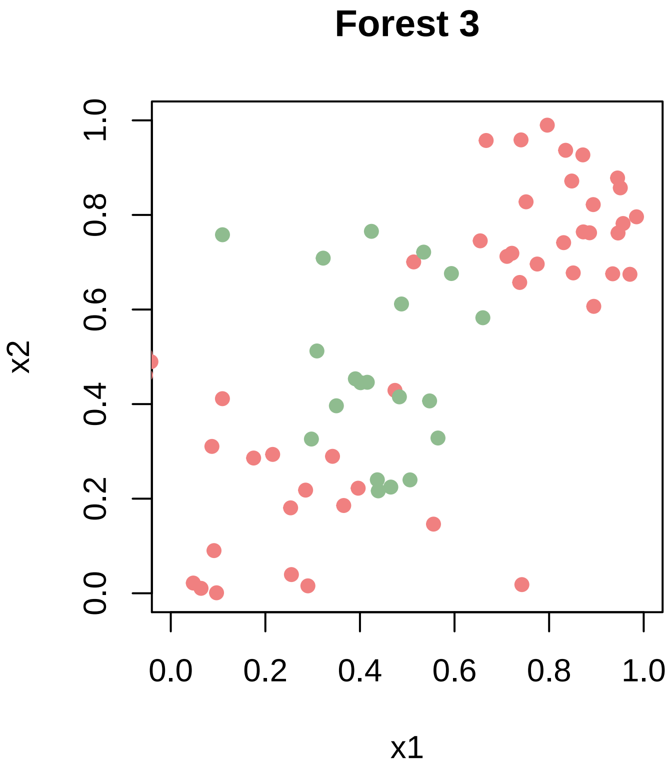
Forest 1 illustrates the problem of finding an **optimal separating hyperplane** for a dataset. In this module you are going to learn a method for finding optimal hyperplanes called **Maximal Margin hyperplanes**.



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. This leads to a classifier called a **Support Vector Classifier** or a **Soft Margin Classifier**.



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.



The reason why we want to separate the data by finding an optimal hyperplane or a separating cruve, is that 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? We classify a new observation based on which side of the decision boundary it falls into.

The methods presented in this module are intended for binary classification, i.e **classification with two classes**, but extensions to more than two classes are briefly mentioned.

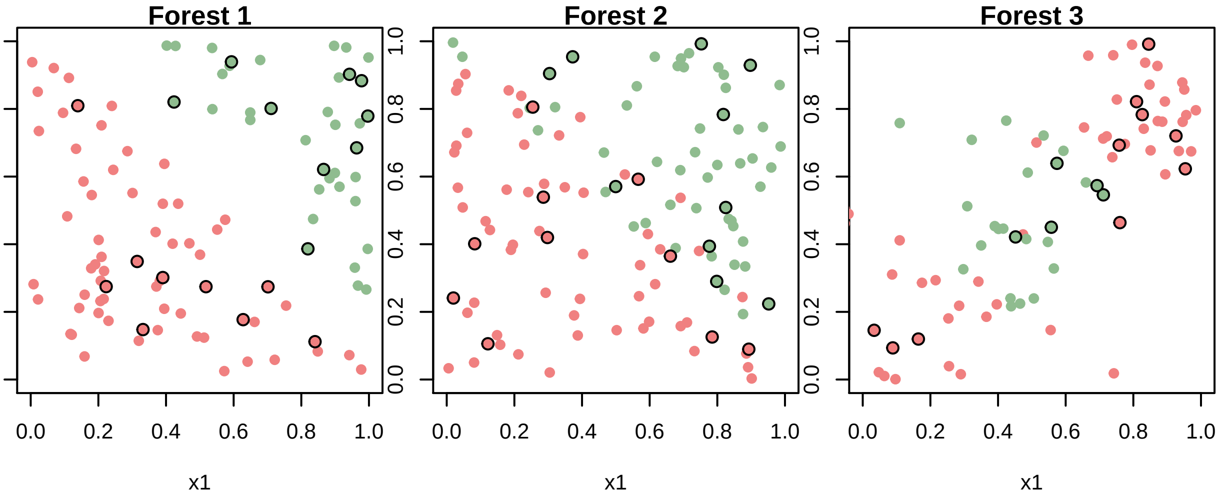
Dataset

The three forests will be used as illustrative examples throughout the module. In this dataset we have two covariates,

the x_1- and the x_2-coordinate of each tree in the forest, and the response is the tree type, either *pine* or *redwood*.

The class *pine* is stored as 1 in our dataset, while *redwood* is stored as -1. (Yes, we have used 0 and 1 before.)

Our goal is 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), and we want to classify the seeds as either pine seeds or redwood seeds given our observations. Thus, the point patterns above can be thought of as the training set, while 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

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

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.

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 ($\sum_{j=1}^p \beta_j^2 = 1$) the value of $\beta_0 + \mathbf{x}^T \boldsymbol{\beta}$ gives the distance from the hyperplane.

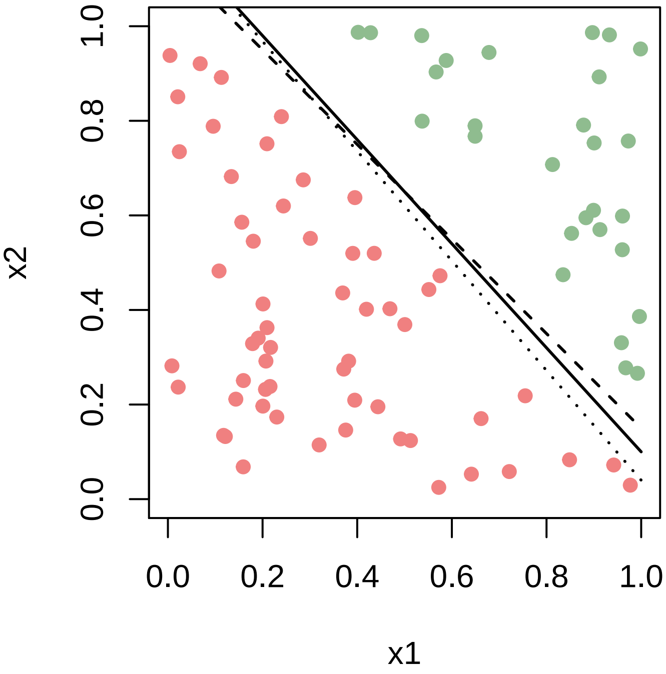
Assumptions

Assume that we have n training observations with p predictors

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

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

Possible hyperplans



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)

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

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.

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:

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

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

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

where M is the width of the margin.

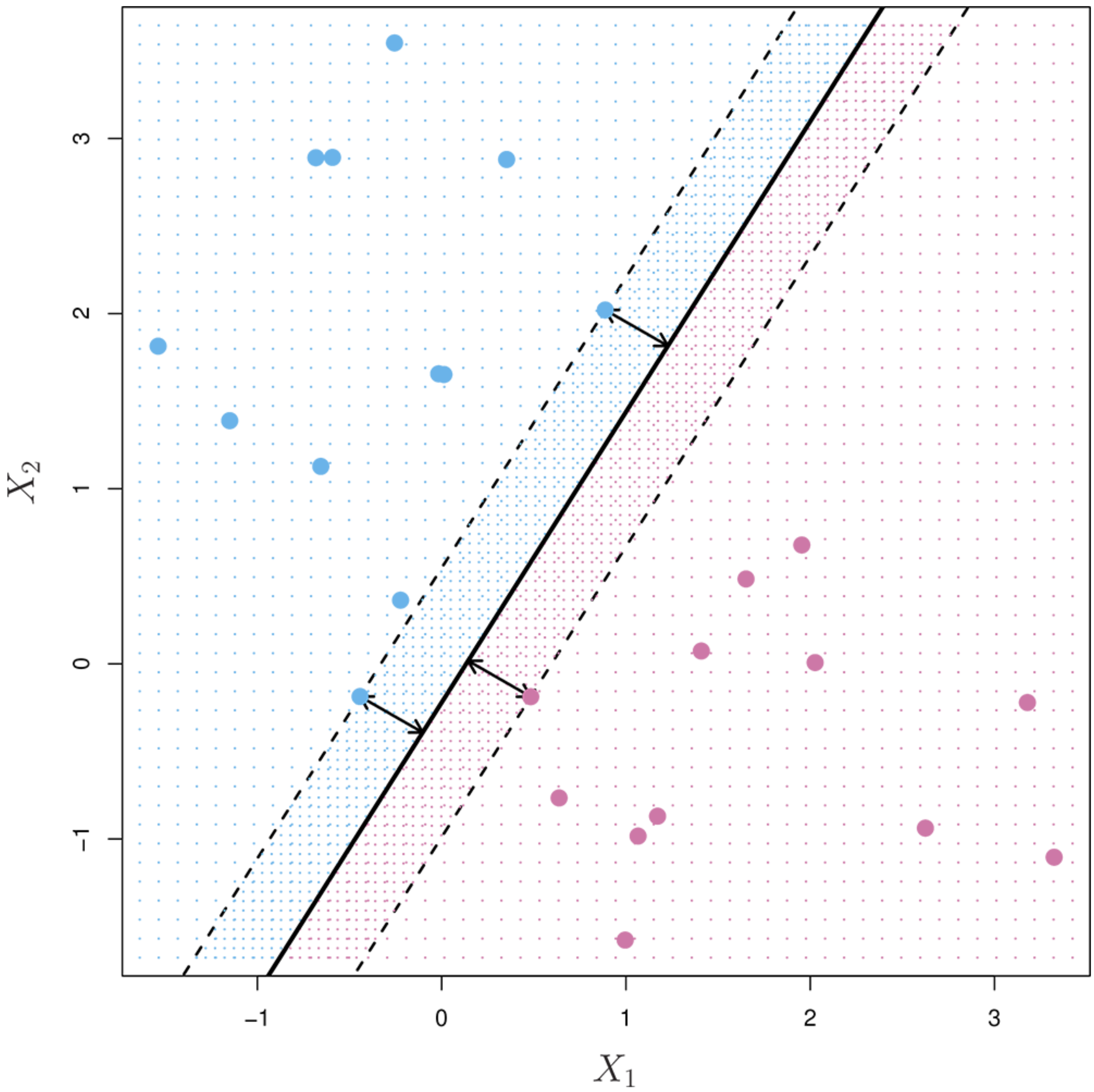


Figure 1: ISLR Figure 9.3

Observe: $y_i(\beta_0 + \mathbf{x}^T\boldsymbol{\beta})$ 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 (Figure 9.3. from James et al. (2013)) 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?
- What if our problem is not separable by a hyperplane?

A:

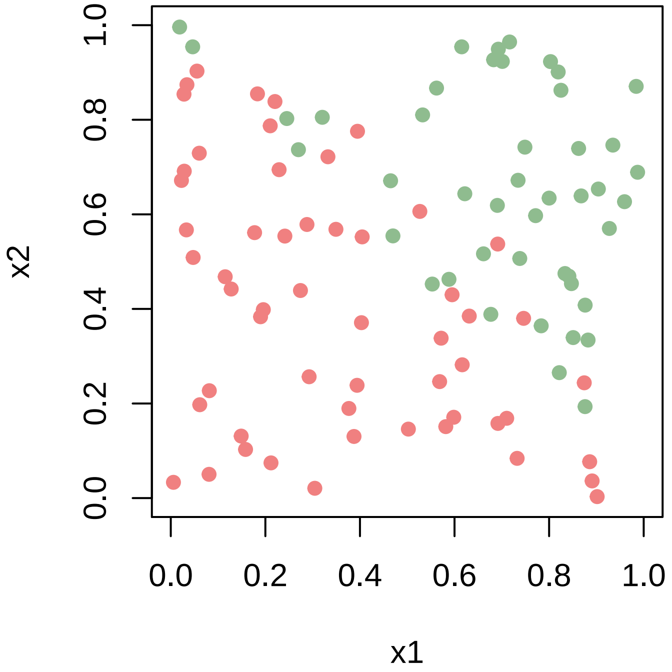
MMC: drawing is the best! Hyperplane with largest possible margin to the support vectors, all training data correctly classified (since separable problem). Only support vectors decide boundary. No distribution assumed for the observations in each class.

No tuning parameter.

Non-separable case is next - by defining slack-variables.

#Support Vector Classifiers

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



It is still possible to construct a hyperplane and use it for classification, but then we have to allow some of the training observations to be misclassified.

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

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

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

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

- 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. No more than C of the observations can be on the wrong side.

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 $f(\mathbf{x}^*) = \beta_0 + \beta_1 x_1^* + ... + \beta_p x_p^*$ 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

- 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?

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

See also Figure 19.3 in Efron and Hastie (2016).

A: Yes, should be standardized because this method treats all variables equally. Same as for lasso and ridge.

If C is small then M must give narrow margin? C is our bias-variance trade-off tuning parameter: C large: allow many violations: more bias, less variance. C small: highly fit the data: less bias, more variance.

##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.

```r
library(e1071)
forest2 = read.table(file = "https://www.math.ntnu.no/emner/TMA4268/2019v/data/forest2.txt")
seeds2 = read.table(file = "https://www.math.ntnu.no/emner/TMA4268/2019v/data/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 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.

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

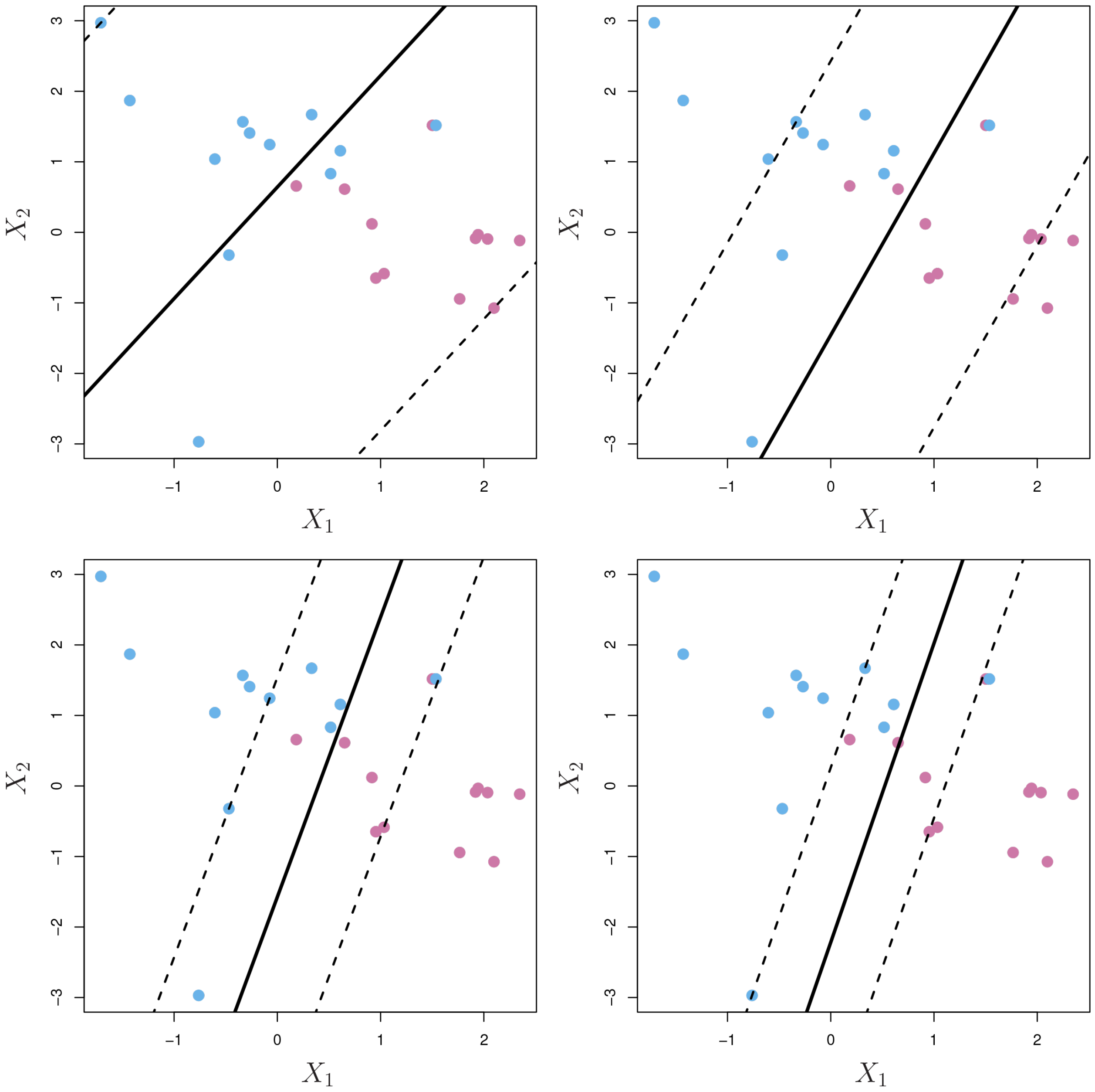
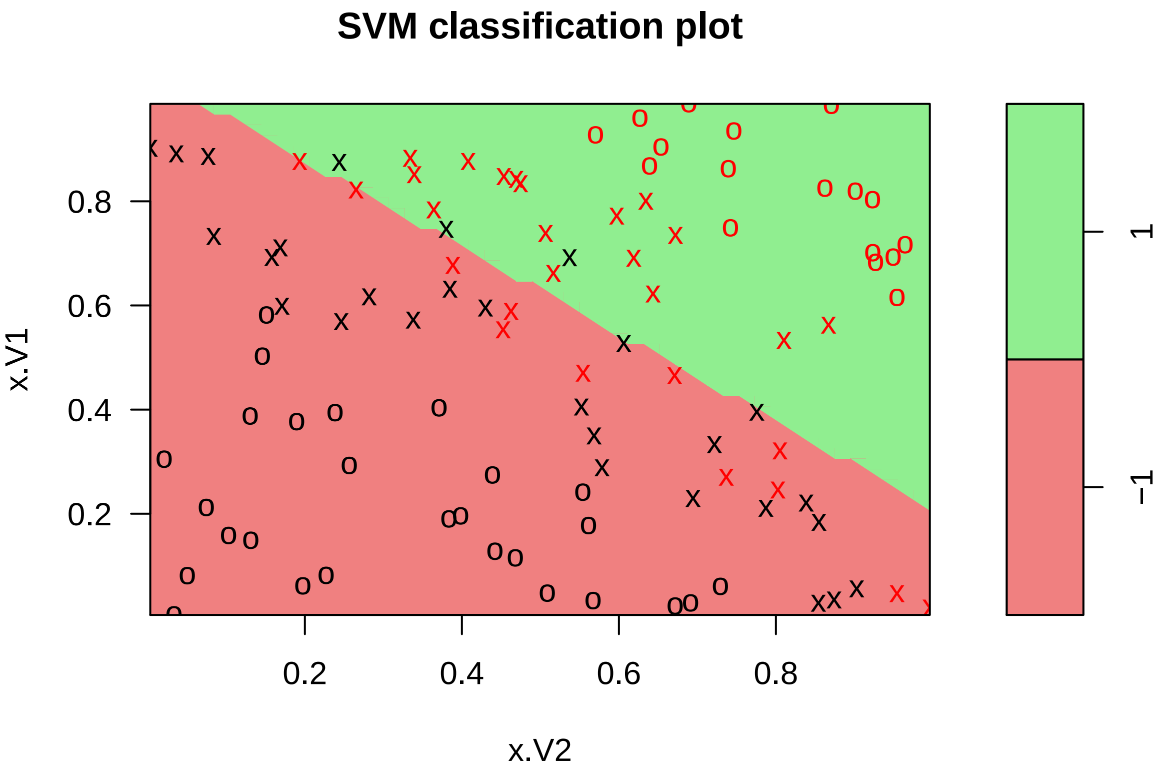


Figure 2: ISLR Figure 9.7: Top left=large C, smaller top right, bottom left and bottom right. As C decreases the tolerance for observations being on the wrong side of the margin decreases and the marin narrows.

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.

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



```
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
```

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

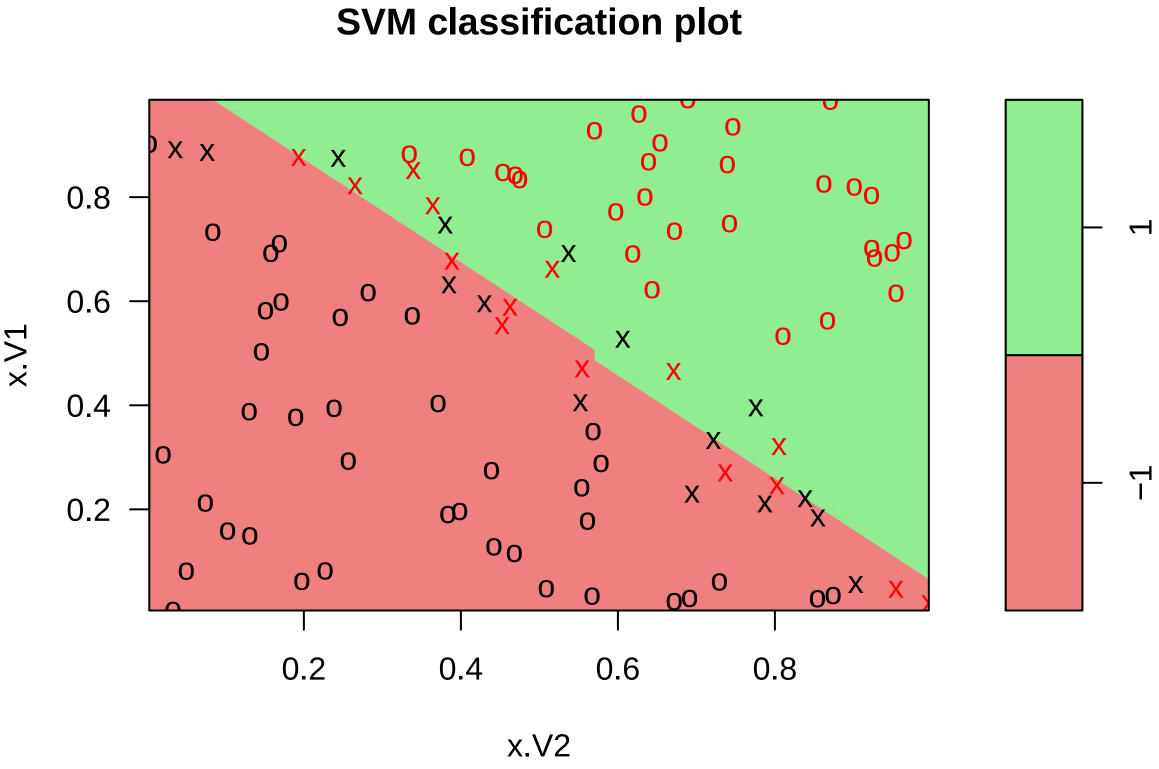
```
##  [1]  1  2  4  6  9 10 16 21 26 27 28 40 44 53 55 57 58 65 67 72 76 77 80
## [24] 81 87 91 92 98  5  8 11 13 18 19 20 23 24 25 34 36 39 41 42 47 48 59
## [47] 61 62 70 71 75 78 88 93 95 96
```

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* $= 100$:

```r
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
```

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(0.001,
    0.01, 0.1, 1, 5, 10, 100)))
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 1e+02  0.15 0.09718253
```

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
```

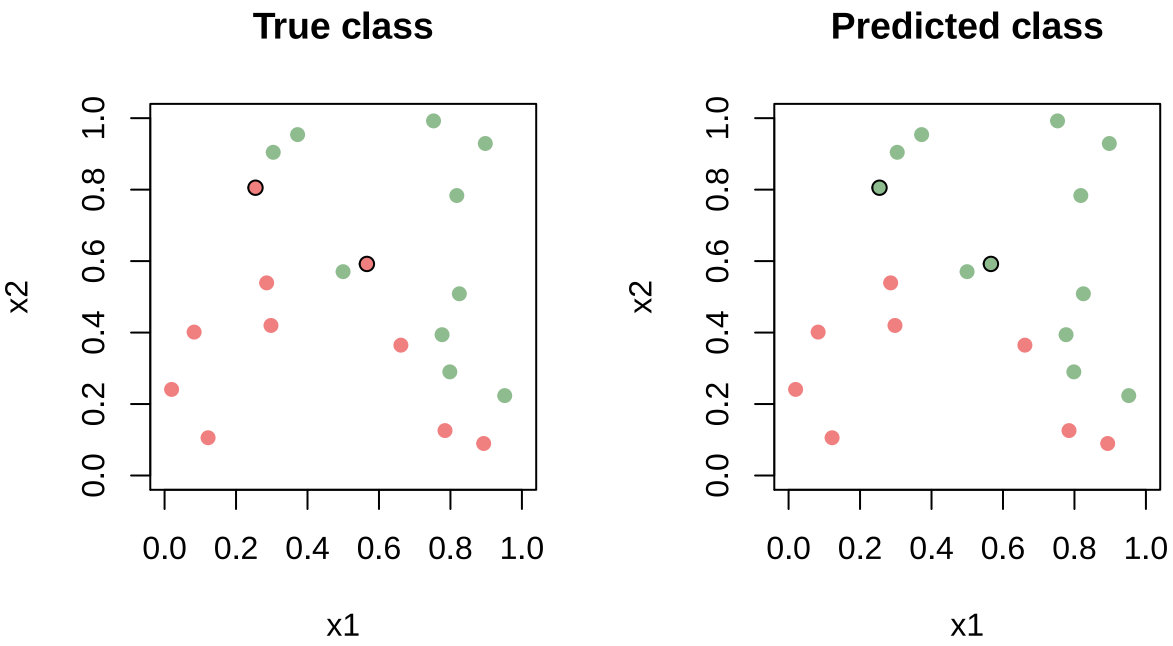
Next, we want to predict the class label of the seeds in the test set. We use the `predict` function and make a confusion table:

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

```
##         truth
## predict -1  1
##      -1  8  0
##      1   2 10
```

```
par(mfrow = c(1, 2))
par(pty = "s")
plot(NA, xlab = "x1", ylab = "x2", xlim = c(0, 1), ylim = c(0, 1))
title("True class")
points(seeds2[seeds2[, 3] == -1, 1:2], pch = 19, col = "lightcoral",
    cex = 0.9)
points(seeds2[seeds2[, 3] == 1, 1:2], pch = 19, col = "darkseagreen",
    cex = 0.9)
points(seeds2[which(ypred_linear != seeds2[, 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(seeds2[ypred_linear == -1, 1:2], pch = 19, col = "lightcoral",
    cex = 0.9)
points(seeds2[ypred_linear == 1, 1:2], pch = 19, col = "darkseagreen",
    cex = 0.9)
points(seeds2[which(ypred_linear != seeds2[, 3]), 1:2], pch = 21)  #Mark misclassification.
```


True class

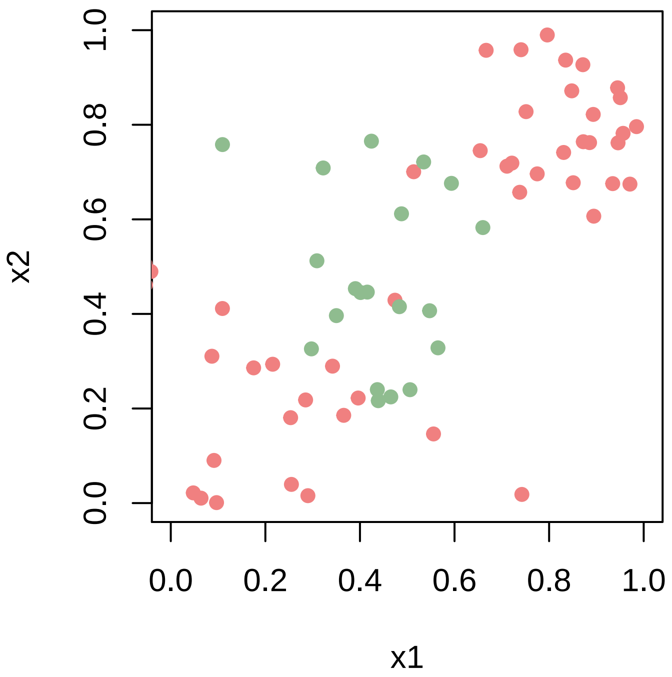
Predicted class

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.



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?

Efron and Hastie (2016) (page 377): *If $n \le p+1$ we can always find a separating hyperplane, unless there are exact features ties across the class barrier.* (Two observations with equal covariate vector, but different classes.)

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. (Right: radial basis function kernel - wait a bit.)

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

Inner products

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.

Remember that we classify a new observation $\mathbf{x}^*$ by first calculating a numerical value for the estimated $f(\mathbf{x}^*)$ and then if $f(\mathbf{x}^*) < 0$ classify as -1 and if $f(\mathbf{x}^*) > 0$ classify as 1.

It can be *shown* (using optimization theory) that the solution to the support vector classifier problem at a new observation $\mathbf{x}^*$, can be expressed as

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

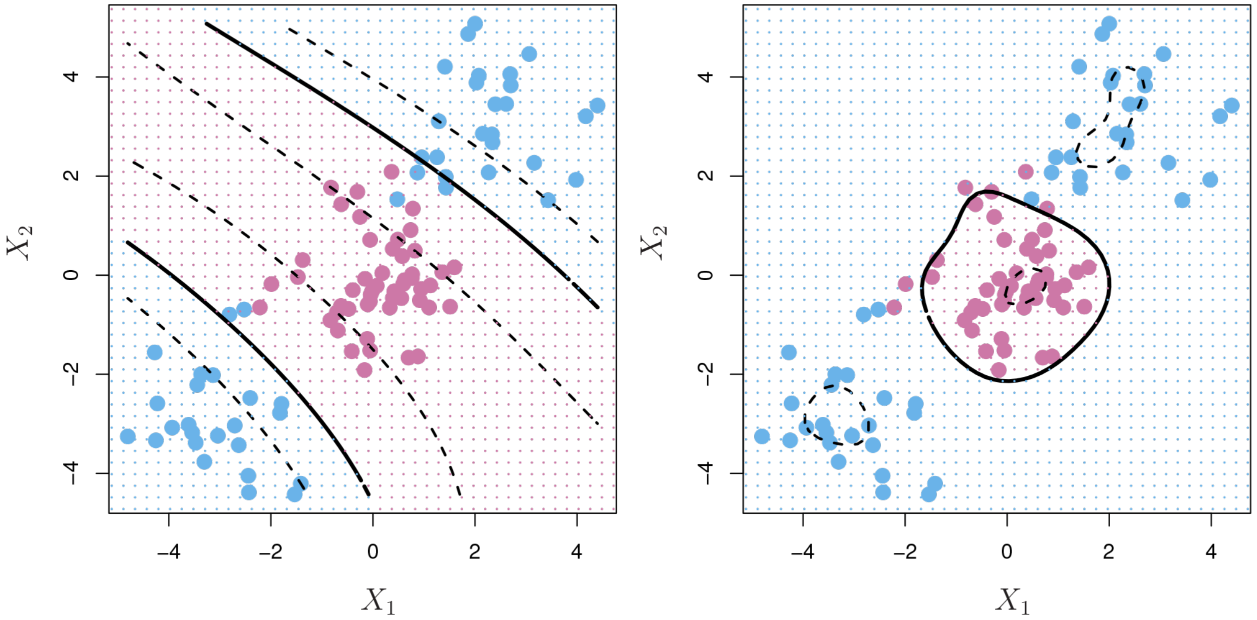


Figure 3: ISLR Figure 9.9

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

A term of the form $\langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle$ denotes the inner product between two observations i and i' and is defined as:

$$\langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}.$$

This means that we need (in addition to the intercept) to estimate n parameters (α_is) instead of p (β_js) (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 $\langle \mathbf{x}_i, \mathbf{x}_i' \rangle$ between all pair of training observations (the class of the training observations is also included).

Also, $\alpha_i = 0$ for the observations i that are *not* the support vectors. Remark: we could alternatively say that $\alpha_i \neq 0$ define the support vectors.

Thus, we only need the inner product between the new observation and the observations corresponding to support vectors to classify a new observation, and

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

where $\mathcal{S}$ contains the indices of the support points. So, we have sparsity in the observations (but not in the predictors).

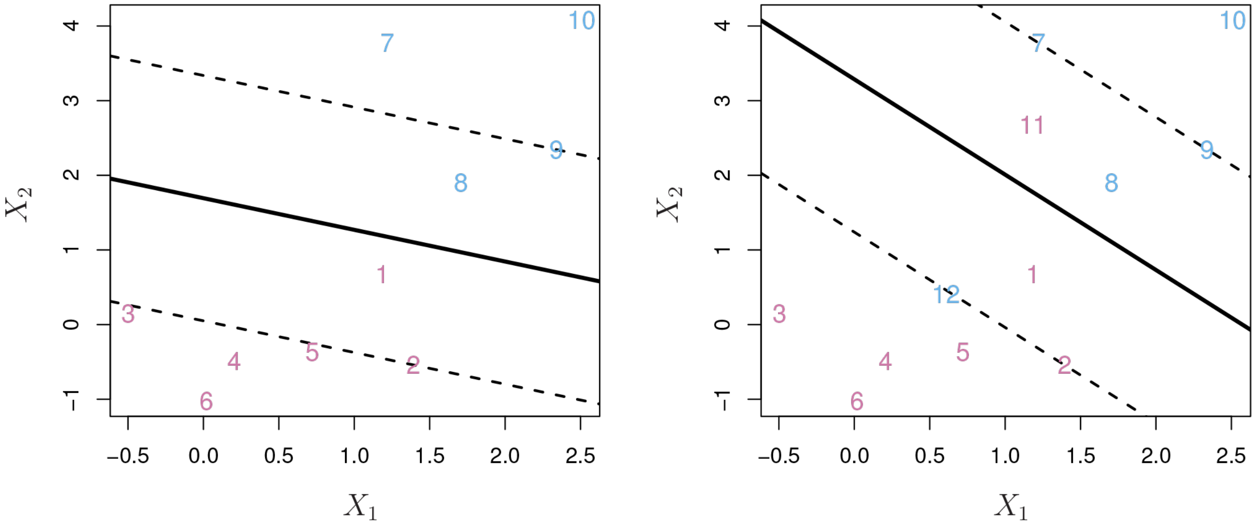


Figure 4: ISLR Figure 9.6

Q: Find the support vectors

Observe: we need all observations (both x and y values) to decide on which observations are the support vectors.

Kernels

(we now use x to denote a new observation)

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

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

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

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:

$$K(\mathbf{x}_i, \mathbf{x}_i') = (1 + \sum_{j1}^p x_{ij} x_{i'j})^d.$$

(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 of the form

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

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

A very popular choice is the radial kernel,

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

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

Observe the connection to a multivariate normal density, 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.

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" (but we have the parameter γ and the budget parameter that we have to decide on).

The radial kernel is convinient if we want a circular decision boundary, and 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.

Computer Age Statistical Inference

Kernels and our optimization

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

$$\text{maximize}_{\beta_0,\alpha_1,\ldots,\alpha_n,\epsilon_1,\ldots,\epsilon_n,} \quad M$$
$$y_i(f(\mathbf{x}_i)) \geq M(1-\epsilon_i) \quad \forall i = 1,...,n.$$
$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n}\epsilon_i \leq C.$$

where

$$f(\mathbf{x}_i) = \beta_0 + \sum_{l\in\mathcal{S}}\alpha_l K(\mathbf{x}_i,\mathbf{x}_l)$$

Tuning parameter example

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

Training errors as ROC and AUC.

Heart data - test error.

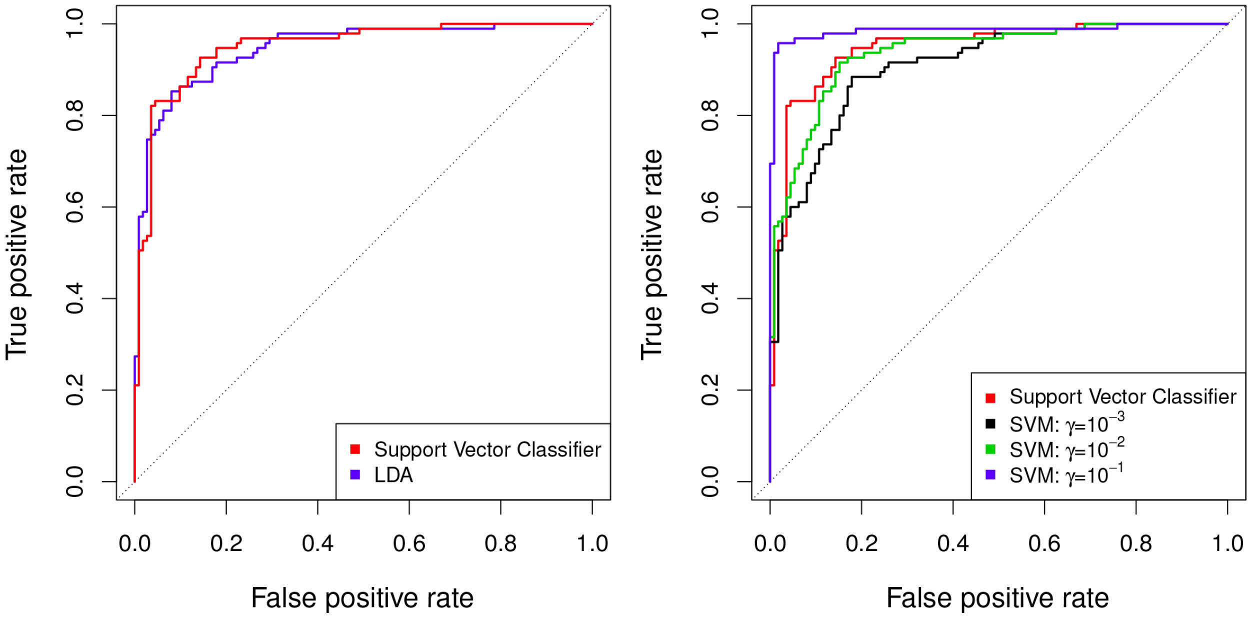


Figure 5: ISLR Figure 9.10

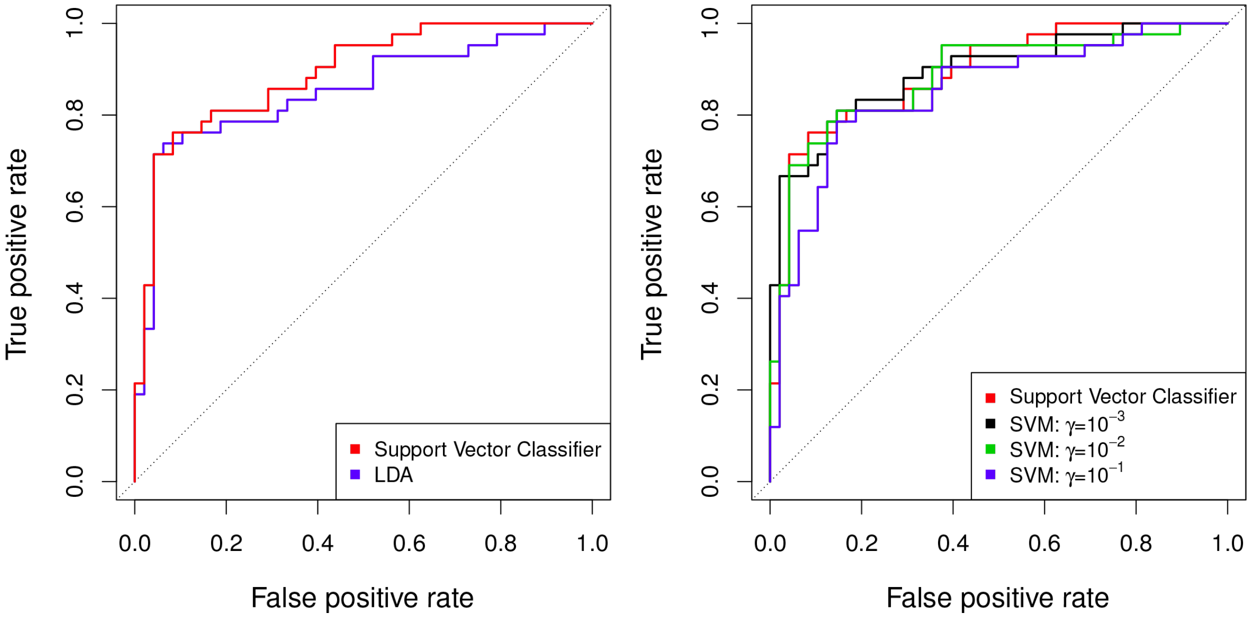


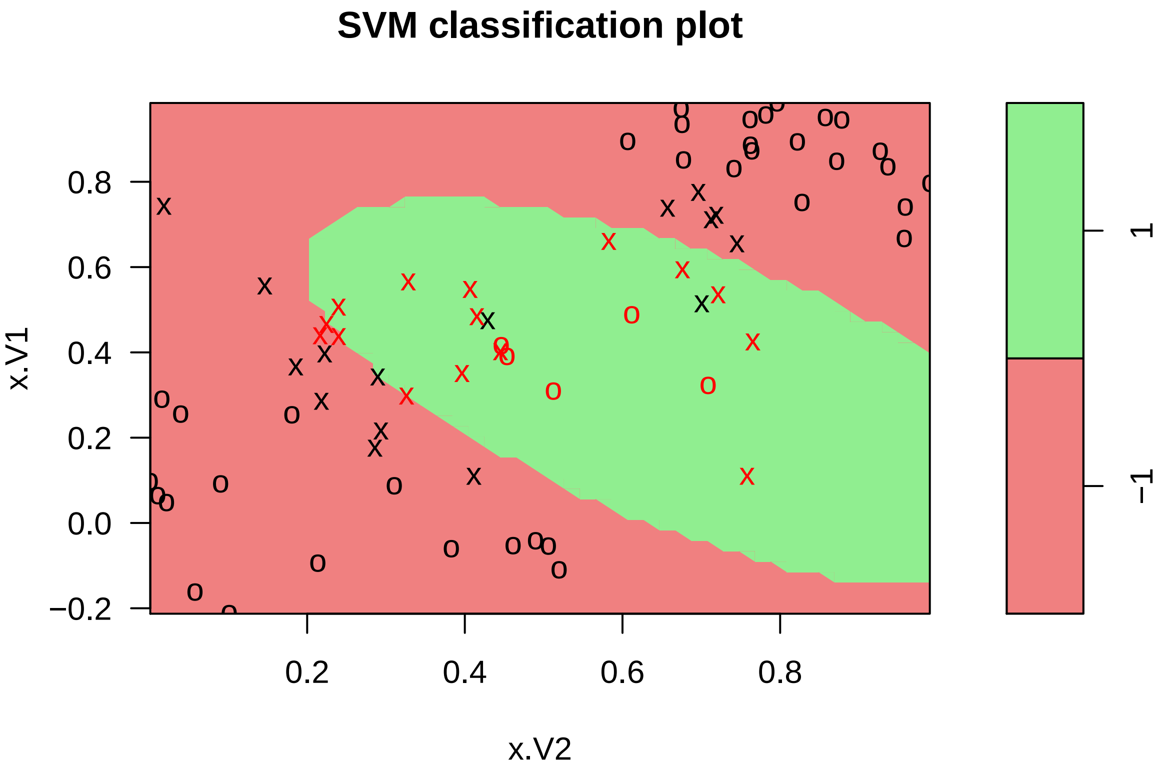
Figure 6: ISLR Figure 9.11

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:

```
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"))
```



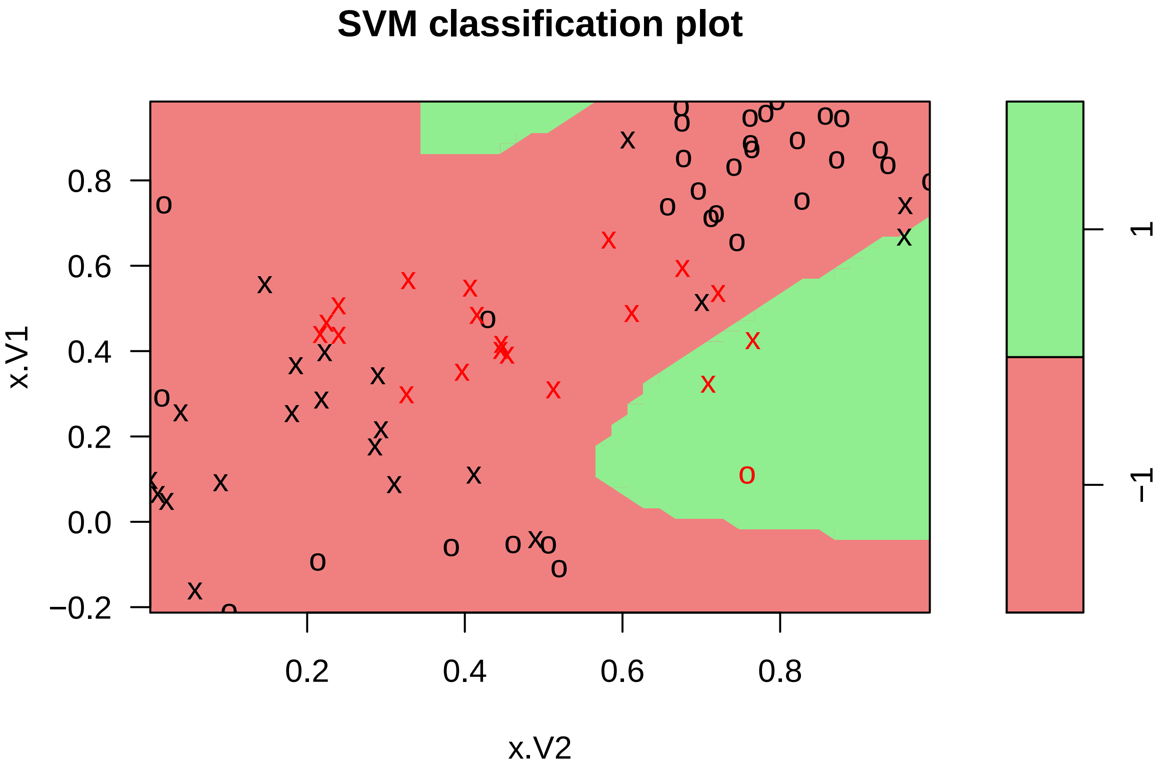
```
summary(svmfit_kernel1)
```

```
##
## Call:
## svm(formula = y ~ ., data = train3, kernel = "radial", gamma = 1,
##     cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10
##
## Number of Support Vectors:  31
##
##  ( 16 15 )
##
##
## 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 = 1e+05, 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
##      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:

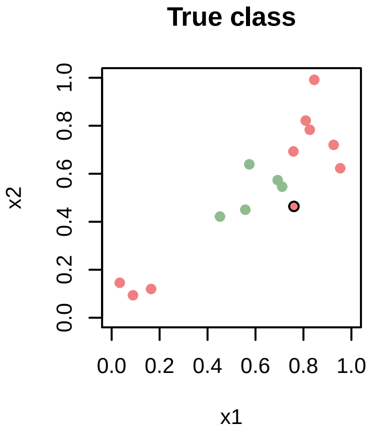
```r
set.seed(1)
CV_kernel = tune(svm, y ~ ., data = train3, kernel = "radial", gamma = 1,
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(CV_kernel)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   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
```

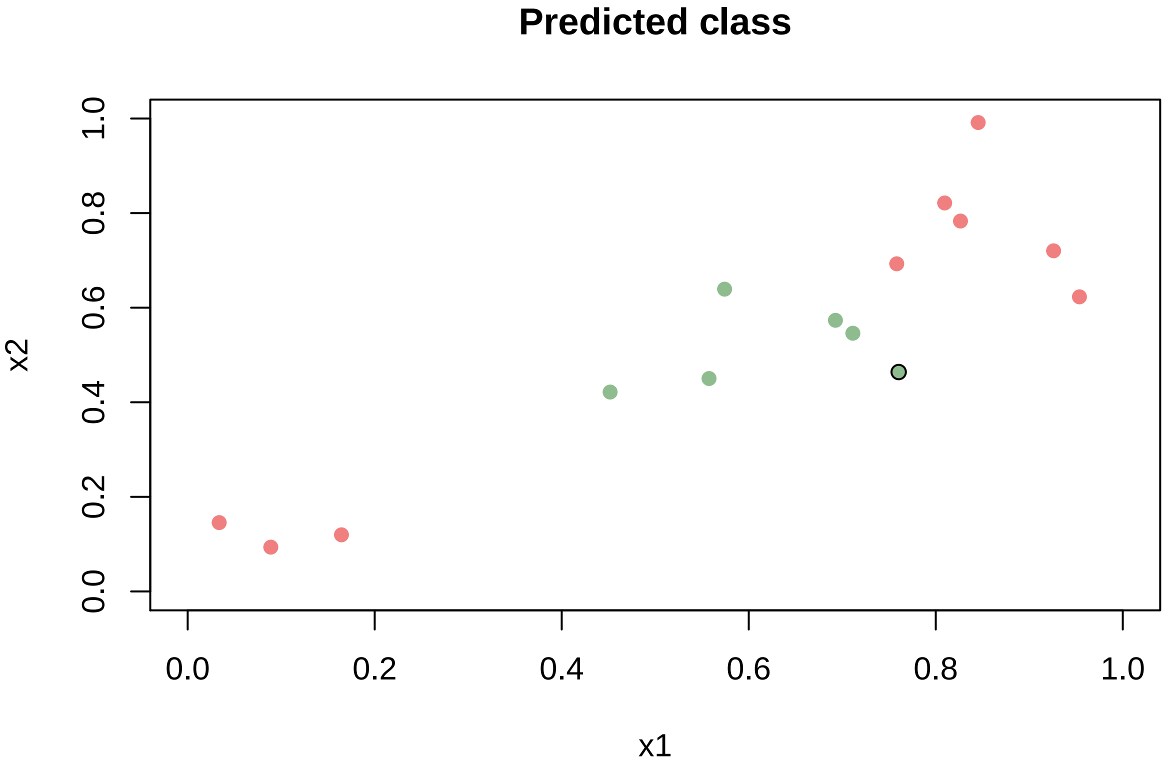
The optimal C is 10. Next, we predict the class label of the seeds in the test set with a model with C=10, make a confusion table and plot the results:

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

```r
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.
```



```r
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.
```



```r
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 $f_k(\mathbf{x})$ where one class is compared to all other classes. Classify a test observation to the class where $f_k(\mathbf{x}^*)$ 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 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

Hinge loss:

$$\max(0, 1 - y_i f(\mathbf{x}_i))$$

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.

Summing up

- 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 (remember ridge-type penalty does not shrink to zero).

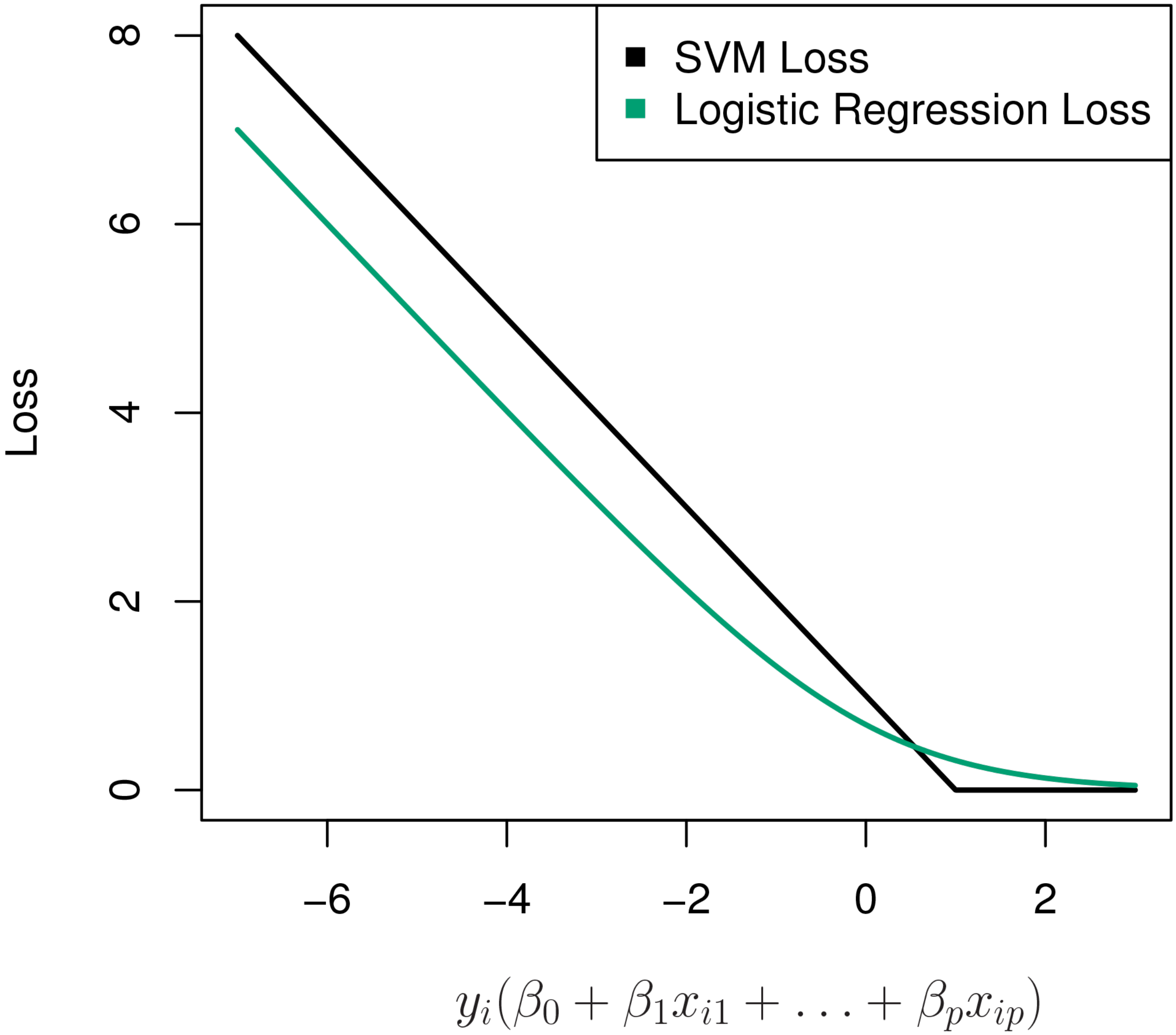


Figure 7: ISLR Figure 9.12: hinge loss - loss 0 for observations on the correct side of the margin

- 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).

Recommended exercises

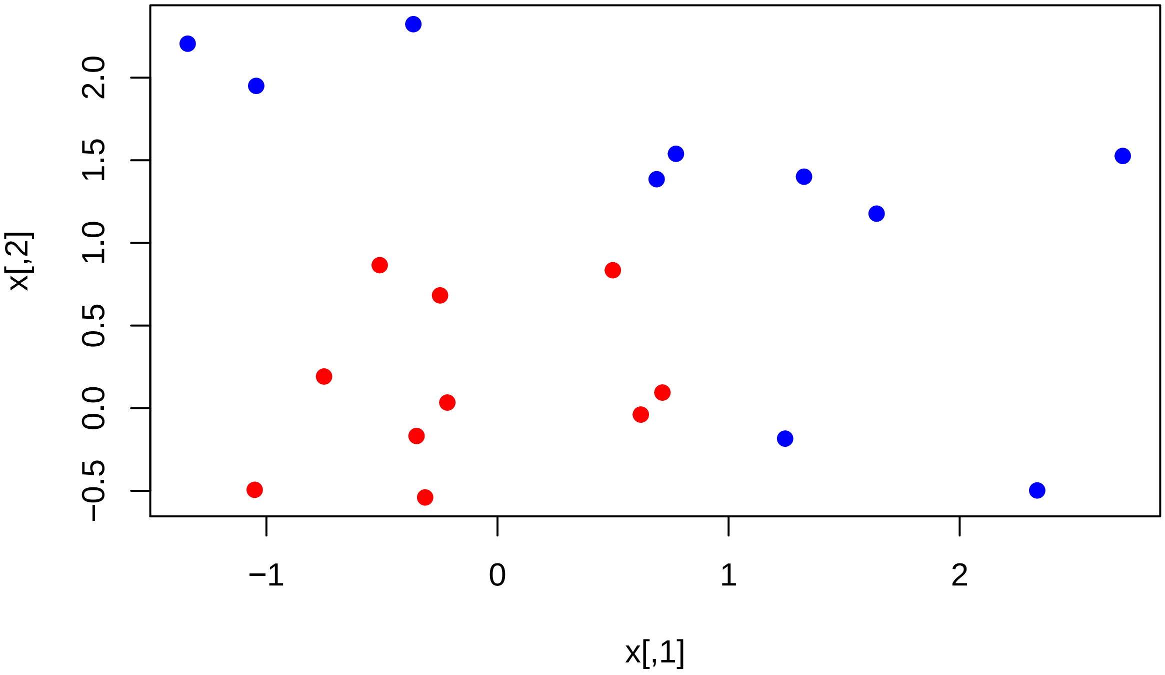
1. Understanding the algorithms:

- 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).

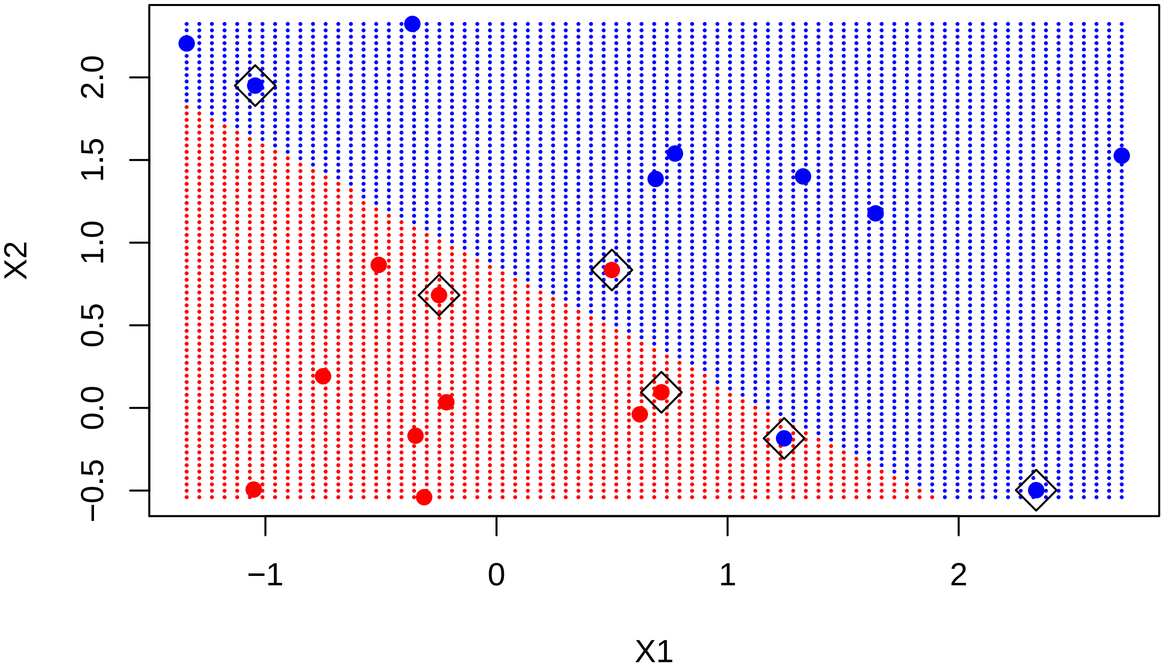
```r
# code taken from video by Trevor Hastie linked above
library(e1071)
# fake data
set.seed(10111)
x = matrix(rnorm(40), 20, 2)
y = rep(c(-1, 1), c(10, 10))
x[y == 1, ] = x[y == 1, ] + 1
plot(x, col = y + 3, pch = 19)
```



```r
# calling svm
dat = data.frame(x, y = as.factor(y))
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
print(svmfit)
```

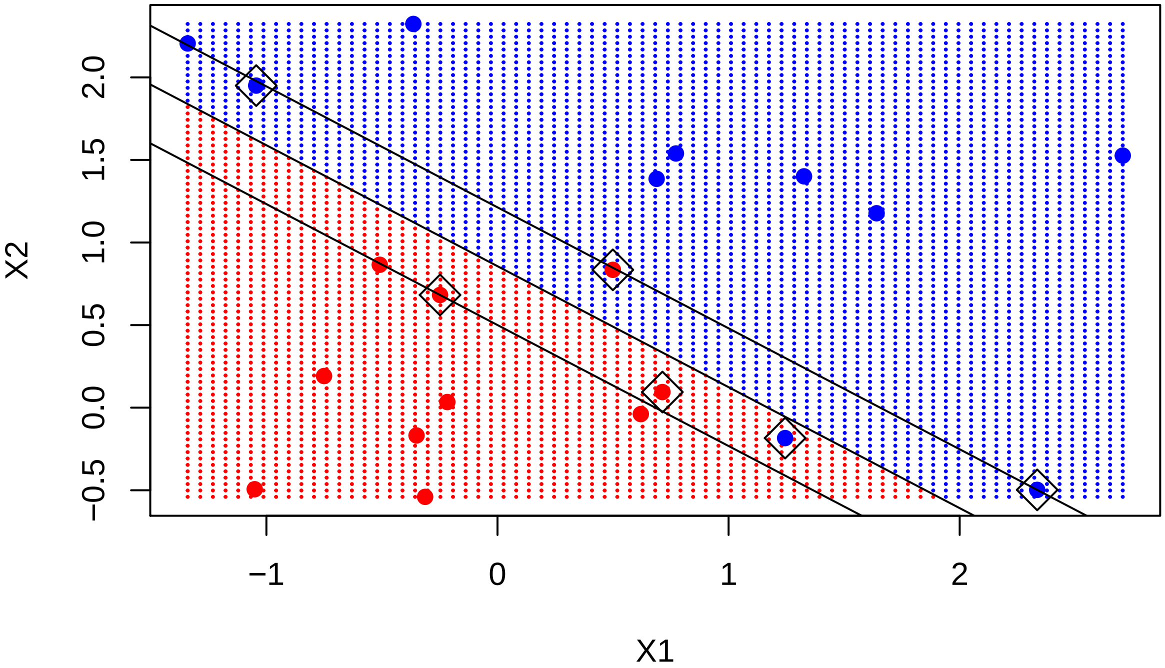
```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  6
```

```
# grid for plotting
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)
    expand.grid(X1 = x1, X2 = x2)
}
xgrid = make.grid(x)
ygrid = predict(svmfit, xgrid)
plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = 0.2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index, ], pch = 5, cex = 2)
```



```
# more info on results - class boundary

beta = drop(t(svmfit$coefs) %*% x[svmfit$index, ])
beta0 = svmfit$rho
plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = 0.2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index, ], pch = 5, cex = 2)
```

```r
abline(beta0/beta[2], -beta[1]/beta[2])  #class boundary
abline((beta0 - 1)/beta[2], -beta[1]/beta[2])  #class boundary-margin
abline((beta0 + 1)/beta[2], -beta[1]/beta[2])  #class boundary+margin
```

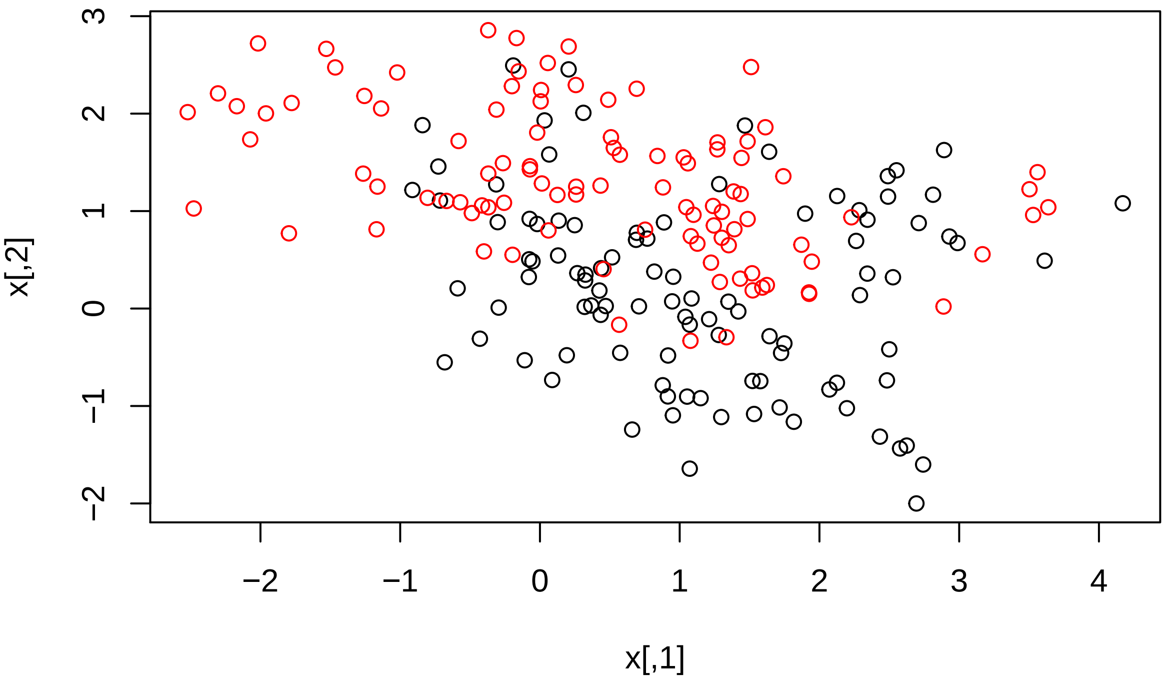


- SVM for non-linear class boundary using simulated data set from Friedman, Hastie, and Tibshirani (2001) where the truth is known (mixtures of normals probably).

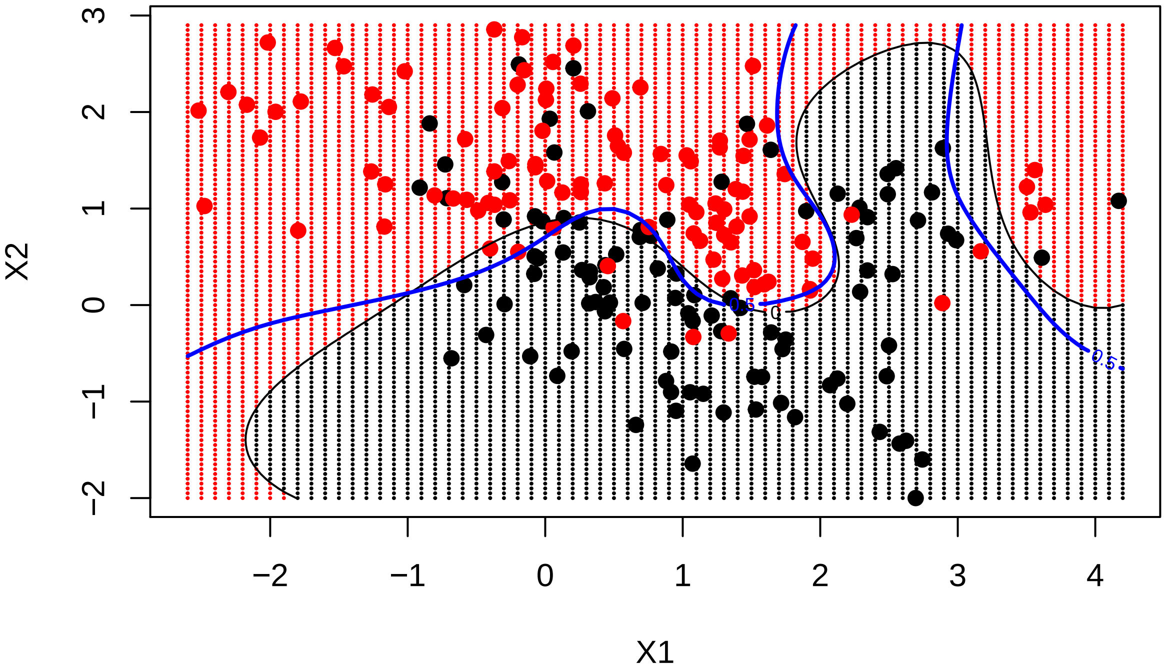
```r
load(url("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda"))
names(ESL.mixture)
```

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

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

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



```
dat = data.frame(y = factor(y), x)
fit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "radial",
    cost = 5)

xgrid = expand.grid(X1 = px1, X2 = px2)
ygrid = predict(fit, xgrid)
plot(xgrid, col = as.numeric(ygrid), pch = 20, cex = 0.2)
points(x, col = y + 1, pch = 19)

# decision boundary
func = predict(fit, xgrid, decision.values = TRUE)
func = attributes(func)$decision
contour(px1, px2, matrix(func, 69, 99), level = 0, add = TRUE)  #svm boundary
contour(px1, px2, matrix(prob, 69, 99), level = 0.5, add = TRUE, col = "blue",
    lwd = 2)  #truth
```

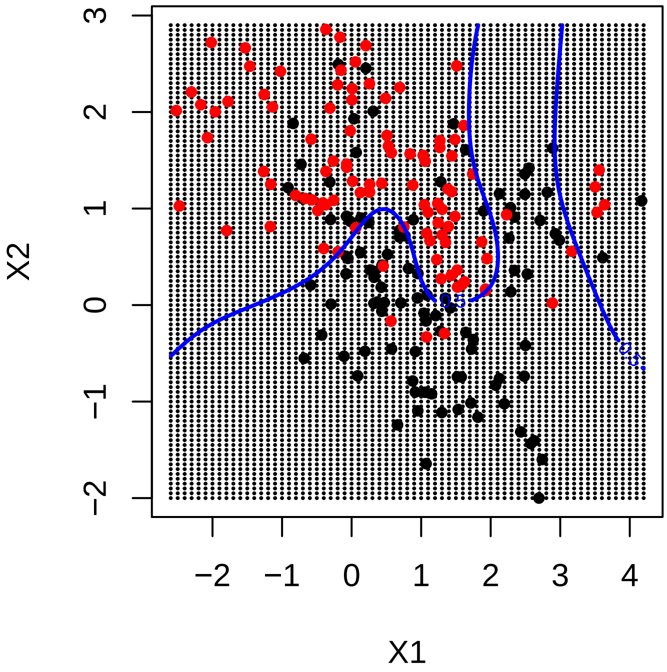


Compulsory exercise 3 2018: Problem 2 - Nonlinear class boundaries and support vector machine

##3 a) Bayes decision boundary [1 point]

We will study classification applied to a simulated data set with two classes from Friedman, Hastie, and Tibshirani (2001), where the data set is supplied together with the Bayes decision boundary. The boundary is plotted below, together with a training set.

```r
load(url("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda"))
# names(ESL.mixture) prob gives probabilites for each class when the
# true density functions are known px1 and px2 are coordinates in x1
# (length 69) and x2 (length 99) where class probabilites are
# calculated
rm(x, y)
attach(ESL.mixture)
dat = data.frame(y = factor(y), x)
xgrid = expand.grid(X1 = px1, X2 = px2)
par(pty = "s")
plot(xgrid, pch = 20, cex = 0.2)
points(x, col = y + 1, pch = 20)
contour(px1, px2, matrix(prob, 69, 99), level = 0.5, add = TRUE, col = "blue",
    lwd = 2)  #optimal boundary
```



- Q11. What is a Bayes classifier, Bayes decision boundary and Bayes error rate? Hint: pages 37-39 in James et al. (2013).
- Q12. When the Bayes decision boundary is known, do we then need a test set?

b) Support vector machine [1 point]

```r
library(e1071)
# support vector classifier
svcfits = tune(svm, factor(y) ~ ., data = dat, scale = FALSE, kernel = "linear",
    ranges = list(cost = c(0.01, 0.1, 1, 5, 10)))
summary(svcfits)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.27
##
## - Detailed performance results:
##    cost error dispersion
## 1  0.01 0.270 0.11595018
## 2  0.10 0.290 0.07745967
## 3  1.00 0.295 0.07619420
## 4  5.00 0.295 0.07619420
## 5 10.00 0.300 0.07817360
```

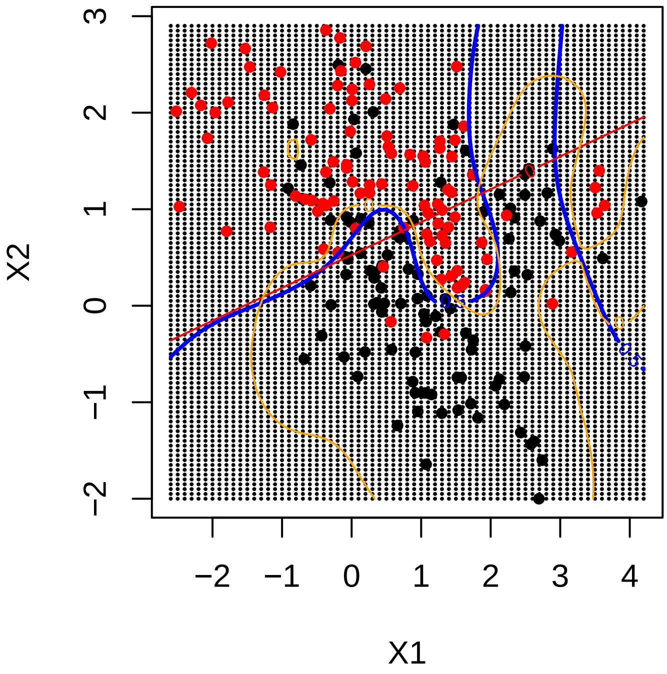
```r
svcfit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "linear",
    cost = 0.01)
```

```r
# support vector machine with radial kernel
set.seed(4268)
svmfits = tune(svm, factor(y) ~ ., data = dat, scale = FALSE, kernel = "radial",
    ranges = list(cost = c(0.01, 0.1, 1, 5, 10), gamma = c(0.01, 1, 5,
        10)))
summary(svmfits)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     5
##
## - best performance: 0.16
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1   0.01  0.01 0.550 0.11547005
## 2   0.10  0.01 0.530 0.12064641
## 3   1.00  0.01 0.275 0.09204468
## 4   5.00  0.01 0.275 0.06346478
## 5  10.00  0.01 0.290 0.08432740
## 6   0.01  1.00 0.535 0.14539219
## 7   0.10  1.00 0.230 0.08232726
## 8   1.00  1.00 0.175 0.06770032
## 9   5.00  1.00 0.175 0.06770032
## 10 10.00  1.00 0.180 0.08881942
## 11  0.01  5.00 0.505 0.20608790
## 12  0.10  5.00 0.400 0.18708287
## 13  1.00  5.00 0.160 0.07745967
## 14  5.00  5.00 0.215 0.09143911
## 15 10.00  5.00 0.195 0.09559754
## 16  0.01 10.00 0.515 0.18566697
## 17  0.10 10.00 0.505 0.18173546
## 18  1.00 10.00 0.190 0.07745967
## 19  5.00 10.00 0.235 0.12483322
## 20 10.00 10.00 0.255 0.13006409
```

```r
svmfit = svm(factor(y) ~ ., data = dat, scale = FALSE, kernel = "radial",
    cost = 1, gamma = 5)

# the same as in a - the Bayes boundary
par(pty = "s")
plot(xgrid, pch = 20, cex = 0.2)
points(x, col = y + 1, pch = 20)
contour(px1, px2, matrix(prob, 69, 99), level = 0.5, add = TRUE, col = "blue",
    lwd = 2)  #optimal boundary

# decision boundaries from svc and svm added
svcfunc = predict(svcfit, xgrid, decision.values = TRUE)
svcfunc = attributes(svcfunc)$decision
```

```r
contour(px1, px2, matrix(svcfunc, 69, 99), level = 0, add = TRUE, col = "red")  #svc boundary
svmfunc = predict(svmfit, xgrid, decision.values = TRUE)
svmfunc = attributes(svmfunc)$decision
contour(px1, px2, matrix(svmfunc, 69, 99), level = 0, add = TRUE, col = "orange")  #svm boundary
```



- Q13. What is the difference between a support vector classifier and a support vector machine?
- Q14. What are parameters for the support vector classifier and the support vector machine? How are these chosen above?
- Q15. How would you evaluate the support vector machine decision boundary compared to the Bayes decision boundary?

R packages

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

```r
install.packages(e1071)
install.packages("knitr")
install.packages("MASS")
```

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.