TMA4268 Statistical Learning V2019

Module 9: SUPPORT VECTOR MACHINES

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

week 11 2019

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.

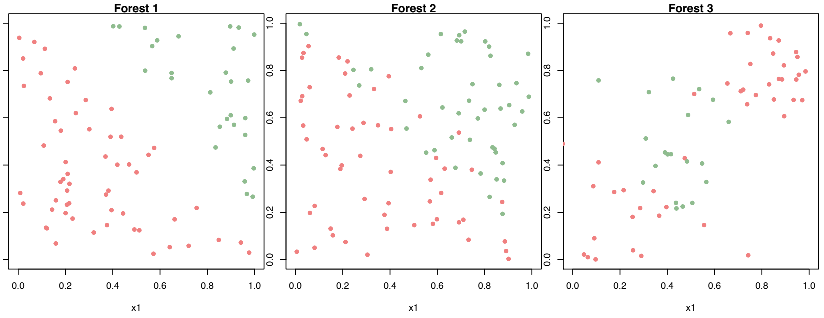
We will only cover classification, and in particular two-class problems.

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



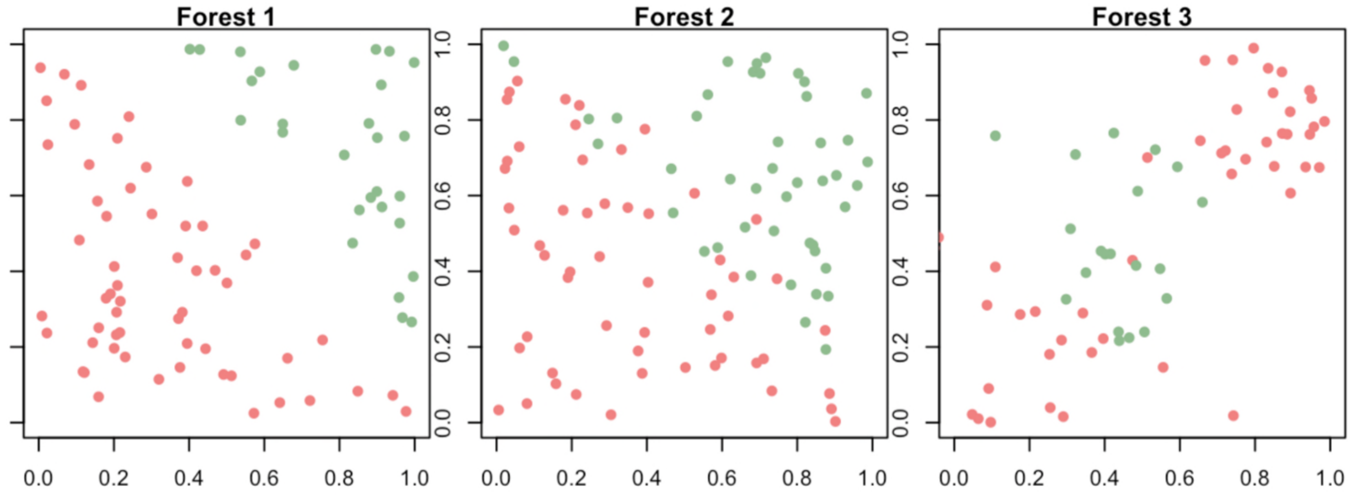
Assume you want to build one continuous fence to separate the two tree types in each of the three study areas. **Where should you build the fence?**

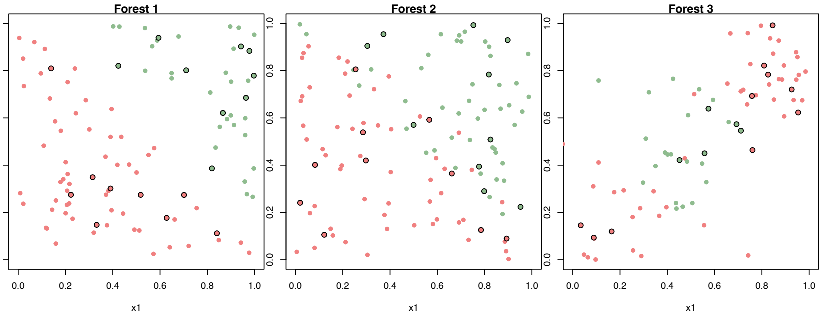
● redwood

● pine

x_1, x_2

position of trees (or seeds)


Forest 1
Forest 2
Forest 3
0.0
0.2
0.4
0.6
0.8
1.0



Forest 1
Forest 2
Forest 3
x1
x1
x1
0.0
0.2
0.4
0.6
0.8
1.0

MAXIMAL MARGIN CLASSIFIER

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.

Maximal Margin Classifier

Hyperplane

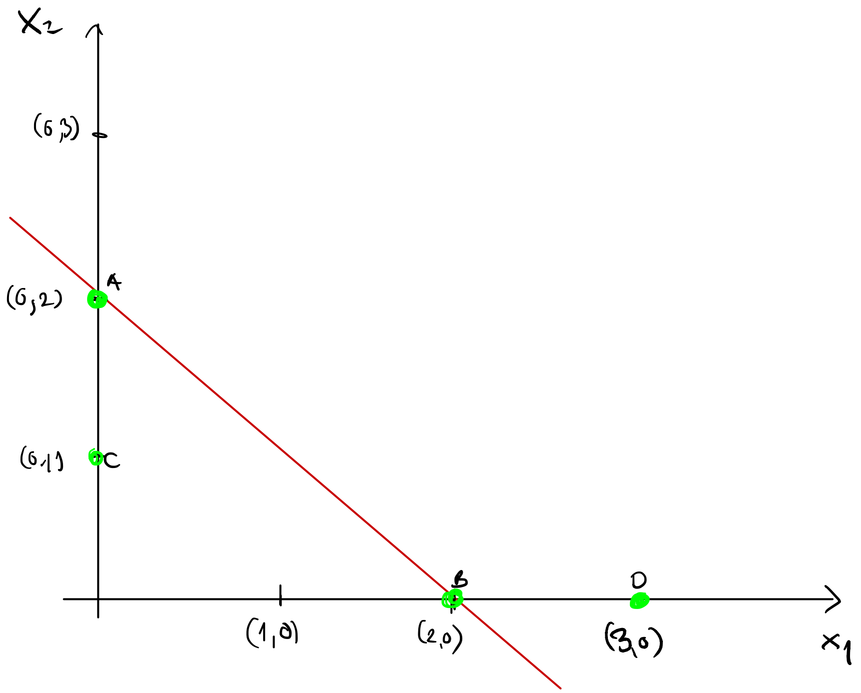
A **hyperplane** in p-dimensions is defined as

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

and is a $p - 1$ dimensional subspace.

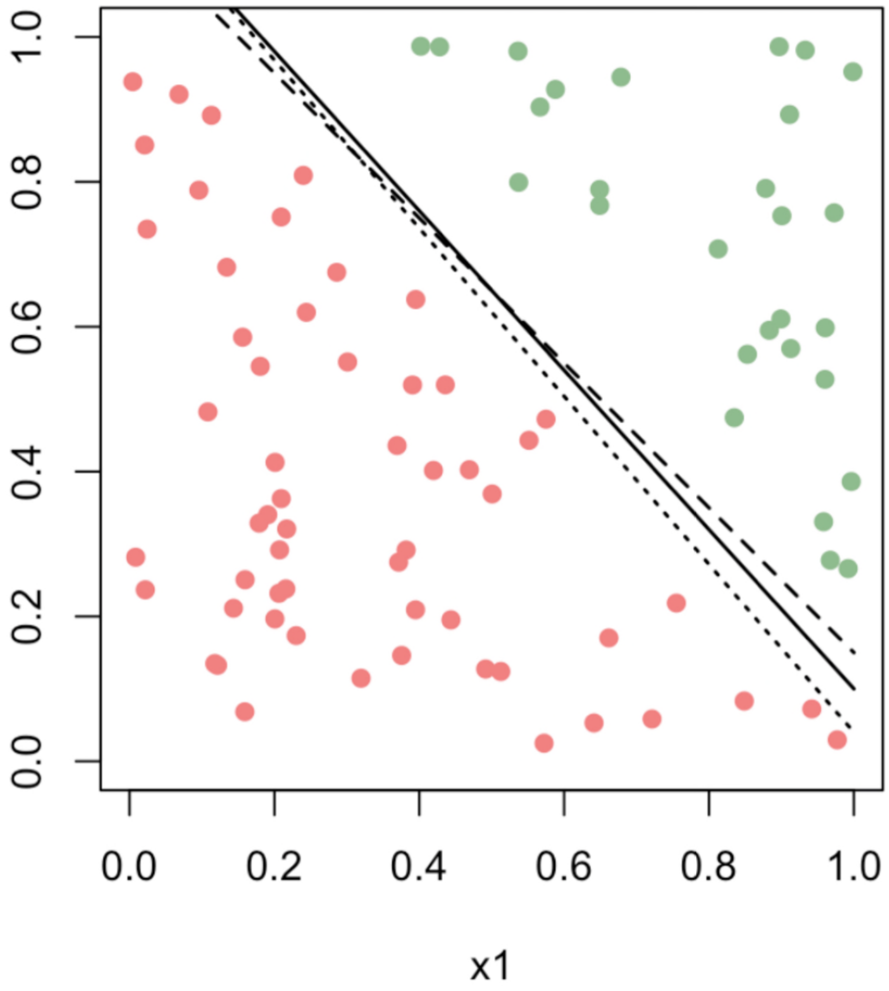
- If a point $X = (X_1, X_2, ..., X_p)^T$ satisfies the above equation, it lies on the hyperplane.
- If $\beta_0 = 0$ the hyperplane goes through the origin (origo).
- The vector $\beta_1, \ldots, \beta_p$ (not including β_0) is called the normal vector and points in the direction orthogonal to the hyperplane.

Example of hyperplane $f(x_1, x_2) = x_1 + x_2 - 2 = 0$

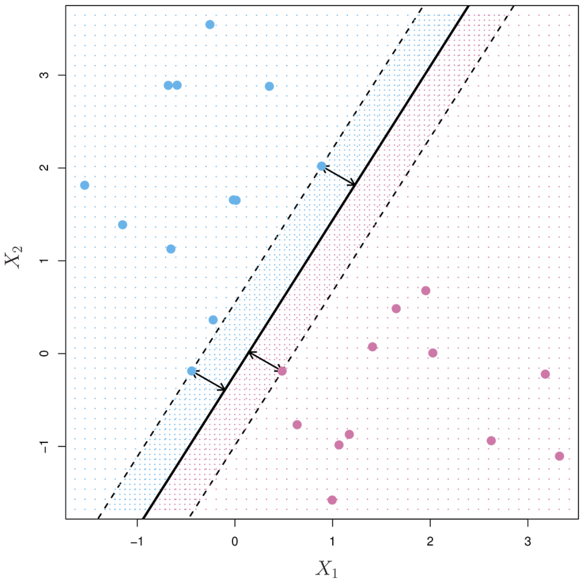


A : (0,2) C : (0,1)

B : (2,0) D : (3,0)



1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x1


X_1
X_2
−1
0
1
2
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:

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

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

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

where M is the width of the margin.

Observe: $y_i(\beta_0 + \mathbf{x}^T\boldsymbol{\beta})$ 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.

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.

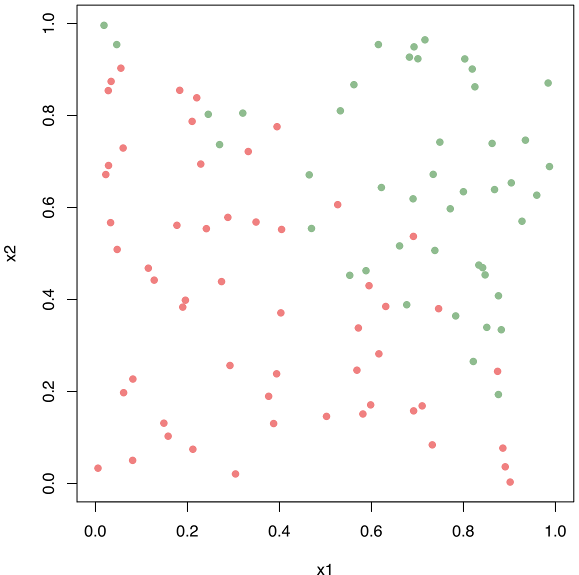
No tuning parameter.

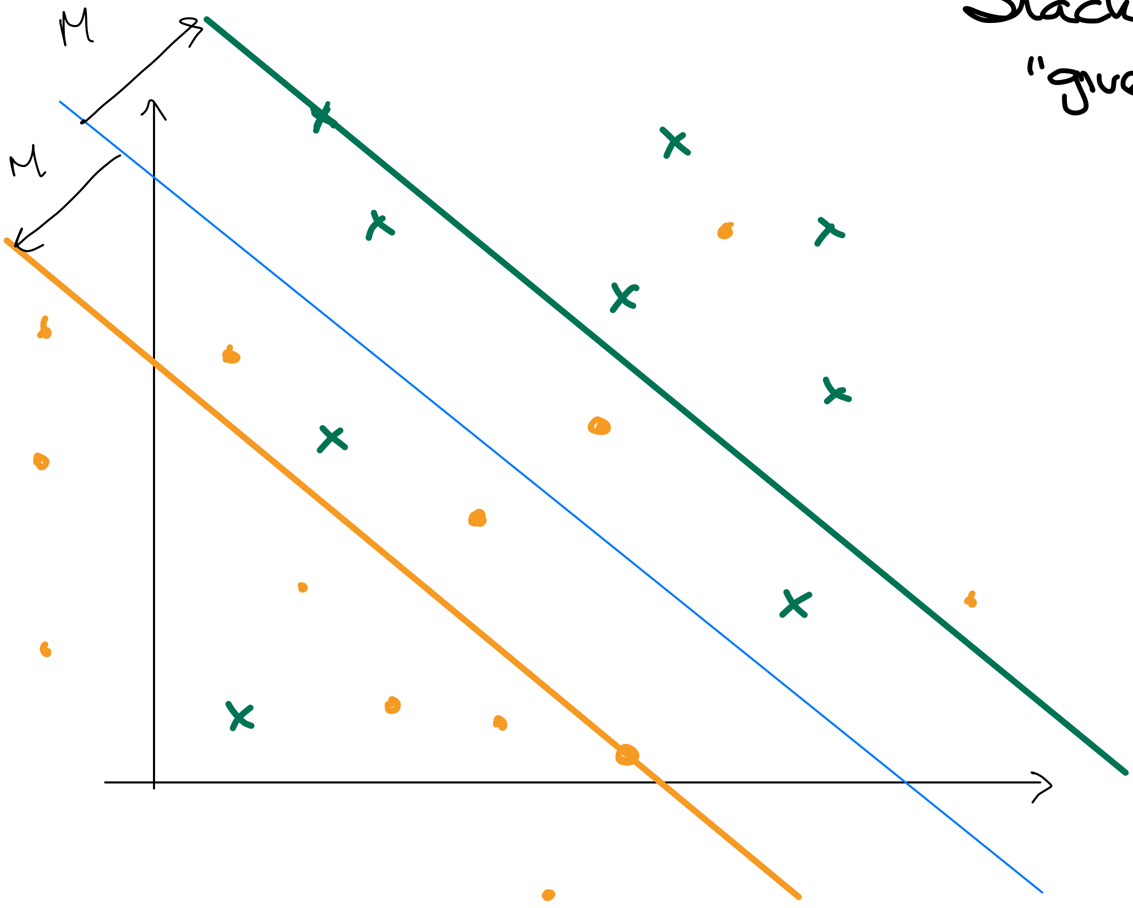
Non-separable case is next - by defining slack-variables.

SUMMING UP

Support Vector Classifiers

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





Slack variable ε_i
"given" to each
observation

Optimization problem

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

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

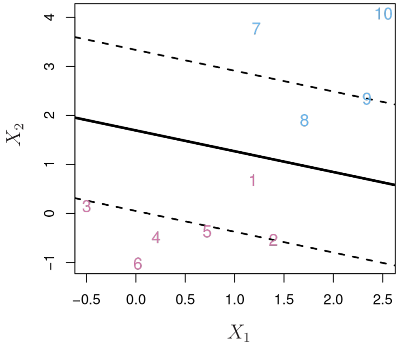
$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C.$$

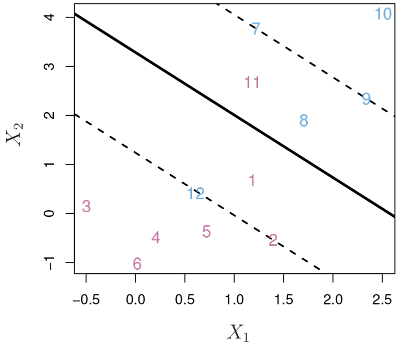
- M is the width of the margin.
- $\epsilon_1, \ldots, \epsilon_n$ are *slack variables*.
  - If $\epsilon_i = 0$ it means that observation i is on the correct side of the margin,
  - if $\epsilon_i > 0$ observation i is on the wrong side of the margin, and
  - if $\epsilon_i > 1$ observation i is on 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 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.

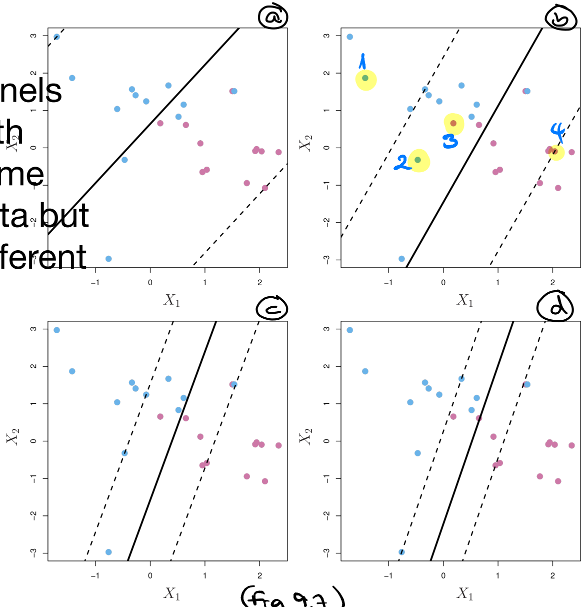
Q: Find the support vectors





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

4 panels with same data but different C



(fig 9.7)

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

$\sum_{i=1}^{n} \varepsilon_i \le C$

$\varepsilon_i = 0$:

$\varepsilon_i > 0$:

$\varepsilon_i > 1$:

WHAT is ε_i for the 4 points?

1: ε_i

2: ε_i

3: ε_i

4: ε_i

Has a, b, c, d largest C? Order abcd?

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.

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.

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).

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/TMA4268/201
seeds2 = read.table(file = "https://www.math.ntnu.no/emner/TMA4268/2019
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.

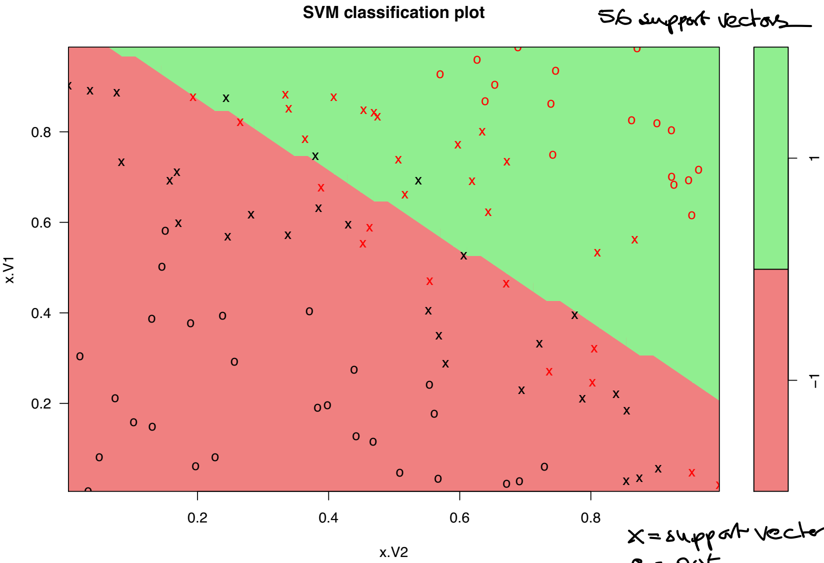
We had in our presentation a budget for errors C, but in `svm` we instead have an argument `cost` that allows us to specify the cost of violating the margin.

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

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"))
```

56 support vectors



x = support vectors
o = not

```
summary(svmfit_linear1)
```

```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
##       gamma:  0.5
##
## Number of Support Vectors:  56
##
##  ( 28 28 )
##
##
## Number of Classes:  2
##
## Levels:
##  -1 1
```

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

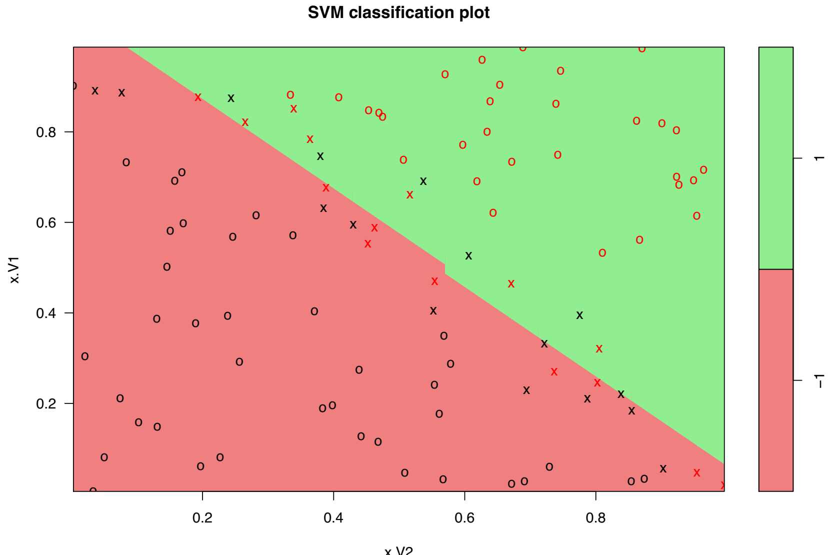
```
##  [1]  1  2  4  6  9 10 16 21 26 27 28 40 44 53 55 57 58 65 67 72 76 77 80
## [24] 81 87 91 92 98  5  8 11 13 18 19 20 23 24 25 34 36 39 41 42 47 48 59
## [47] 61 62 70 71 75 78 88 93 95 96
```

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:

```
svmfit_linear2 = svm(y ~ ., data = train2, kernel = "linear", cost = 10
    scale = FALSE)
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(0.001,
    0.01, 0.1, 1, 5, 10, 100)))
summary(CV_linear)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.15
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-03  0.45  0.1779513
## 2 1e-02  0.22  0.1751190
## 3 1e-01  0.15  0.1269296
## 4 1e+00  0.15  0.1269296
## 5 5e+00  0.15  0.1080123
## 6 1e+01  0.15  0.1080123
## 7 1e+02  0.15  0.1080123
```

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:

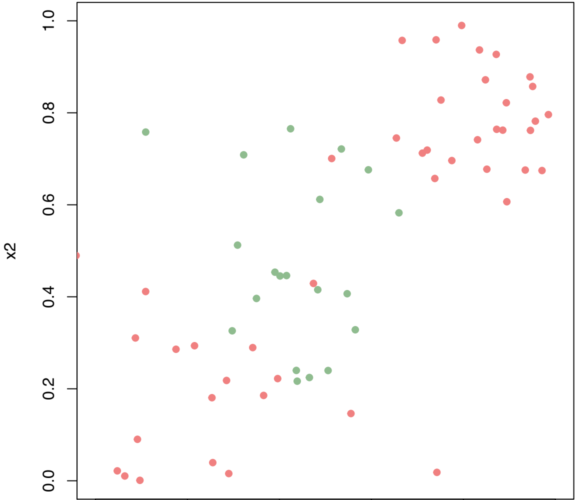
```
ypred_linear = predict(bestmod_linear, test2)
table(predict = ypred_linear, truth = test2[, 3])
```

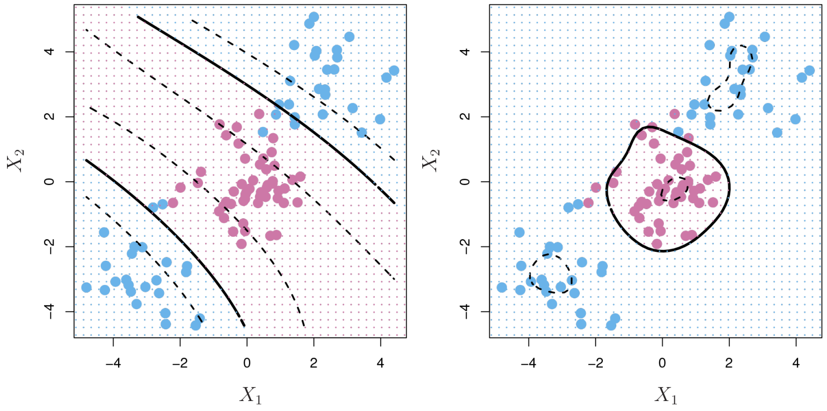
```
##          truth
## predict -1 1
##      -1  9 2
##      1   1 8
```

SUMMING UP

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.





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.

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 373): *If $n \geq p+1$ 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.)

Beck to SVC: $f(x) = \beta_0 + x^T\beta$ class boundery linear

RESULT :

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).

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:

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

(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 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).

RADIAL KERNEL

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

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).

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

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

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

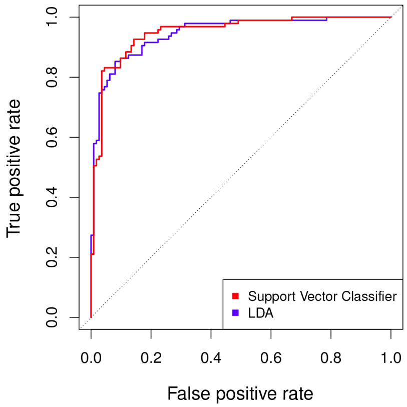
where

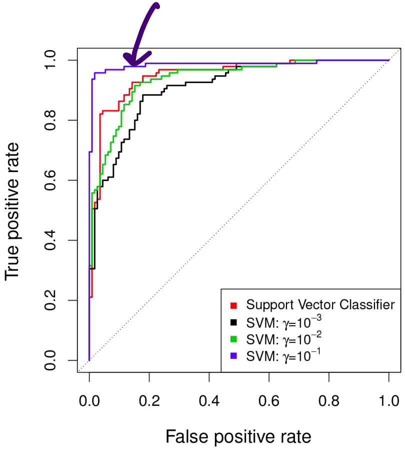
$$f(\mathbf{x}_i) = \beta_0 + \sum_{l \in \mathcal{S}} \alpha_l K(\mathbf{x}_i, \mathbf{x}_l)$$

Tuning parameter example → important to avoid overfitting

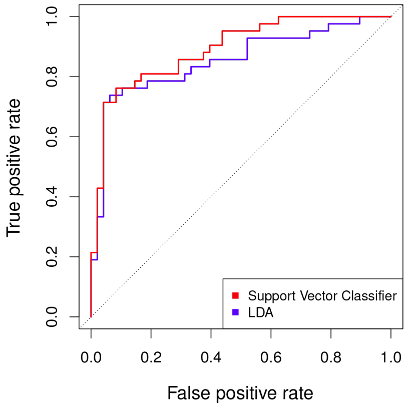
Heart data - predict heart disease from $p = 13$ predictors.

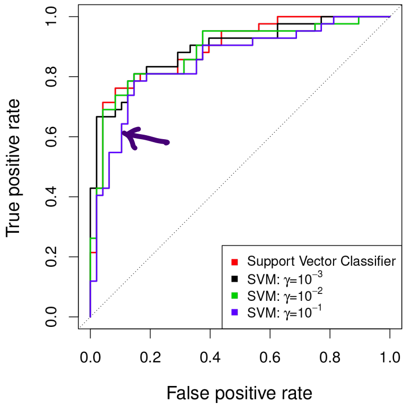
Training errors as ROC and AUC.

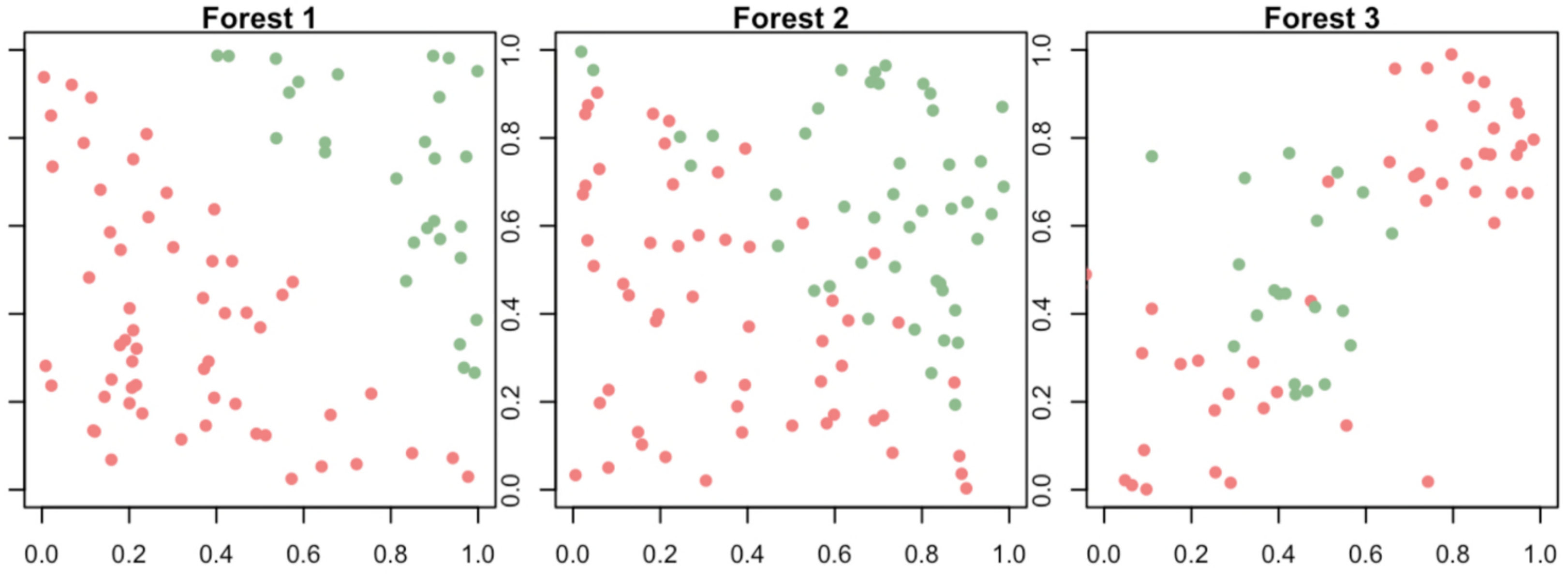




Heart data - test error.



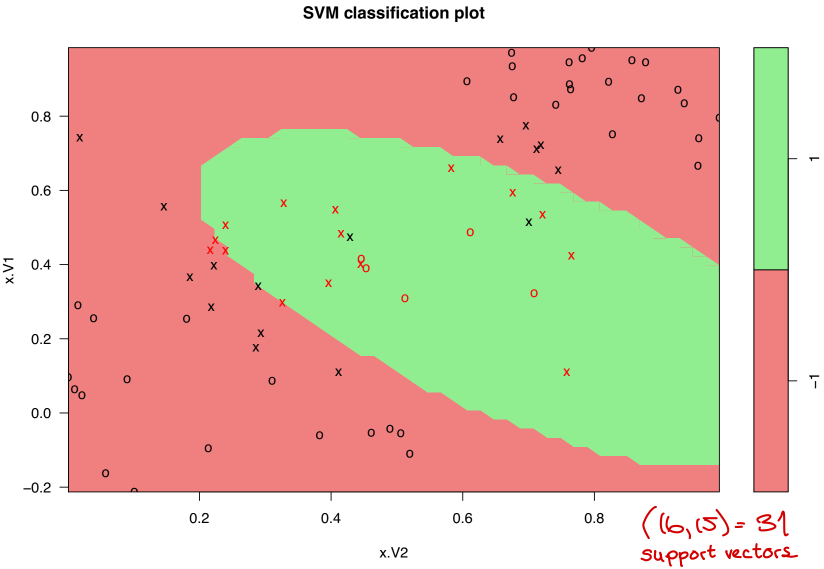




TRY RADIAL BASIS SVM

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

$\gamma = 1.0$



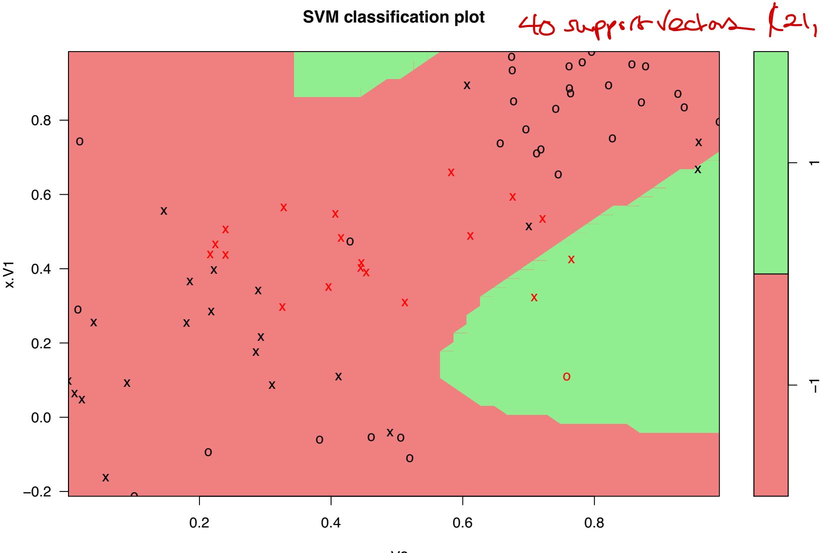
$(16, 15) = 31$ support vectors

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

=4

```
svmfit_kernel2 = svm(y ~ ., data = train3, kernel = "polynomial", degre
    cost = 1e+05, scale = FALSE)
plot(svmfit_kernel2, train3, col = c("lightcoral", "lightgreen"))
```

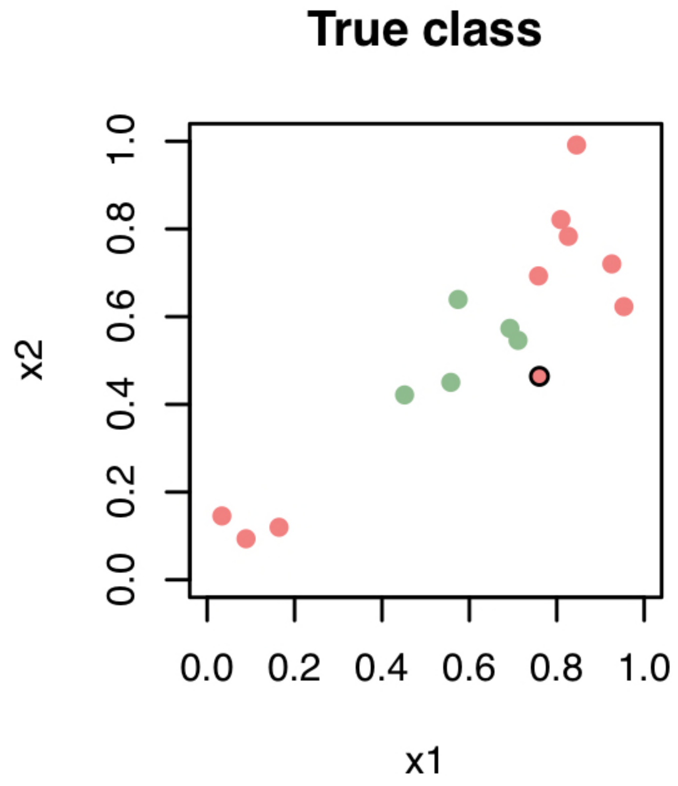
40 support vectors (21, 19)

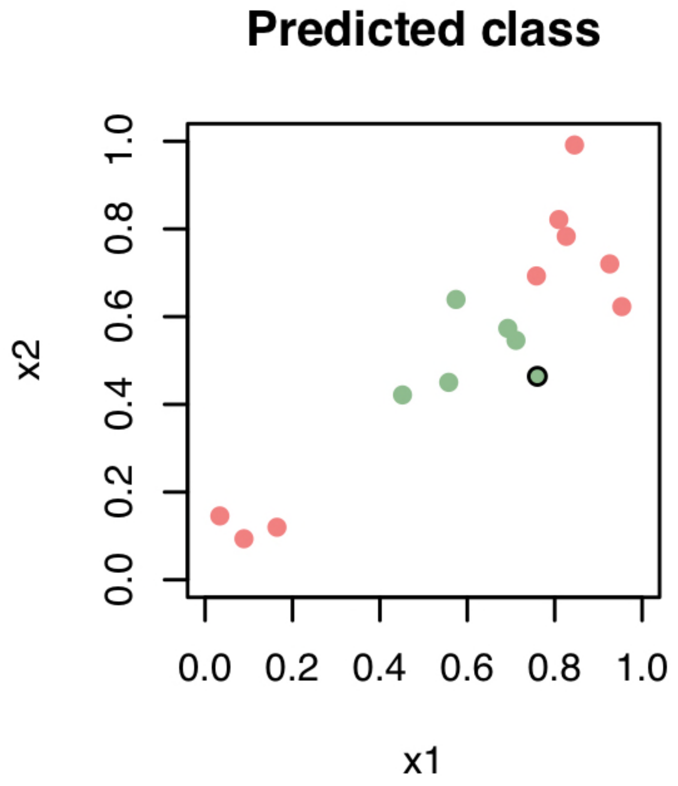


For this dataset a radial kernel is a natural choice: A circular decision boundary seems like a good idea. Thus, we proceed with `kernel='radial'`, and use the `tune()` function to find the optimal tuning parameter C:

```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
##    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
```





10 seeds of each
of redwood and pine

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 (svc) 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

Hinge loss:

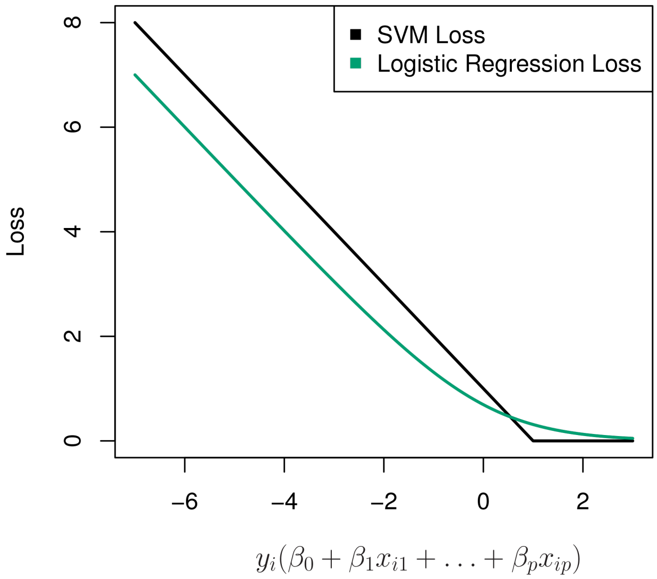
$$\max(0, 1 - y_i f(\mathbf{x}_i))$$

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.


■ SVM Loss
■ Logistic Regression Loss
Loss
0
2
4
6
8
−6
−4
−2
0
2
$y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip})$

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).
- 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)
```