

06. Support Vector Machines

Table of Contents

- ① Introduction to support vector machines
- ② Maximal margin classifier
- ③ Support vector classifier
 - Margins and slack variables
 - Non-linear decision boundaries
- ④ Support vector machines (SVM)
 - Linear kernel
 - Polynomial kernel
 - Radial kernel

Support Vector Machines

- Here we approach the two-class classification problem in a direct way:
 - We try and find a plane that separates the classes in feature space.
- If we cannot, we get creative in two ways:
 - We soften what we mean by “separates”, and
 - We enrich and enlarge the feature space so that separation is possible.
- Three methods
 - Maximal margin classifier
 - Support vector classifier (soft margin classifier)
 - Support vector machines

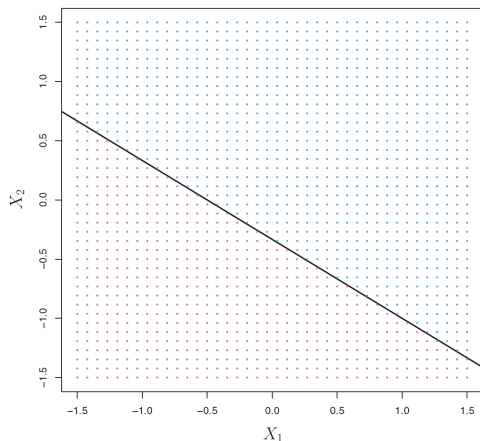
What is a Hyperplane?

- A **hyperplane** in p dimensions is a flat affine subspace of dimension $p - 1$.
- In general the equation for a hyperplane has the form

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0 \quad (1)$$

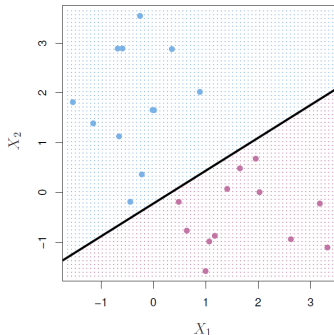
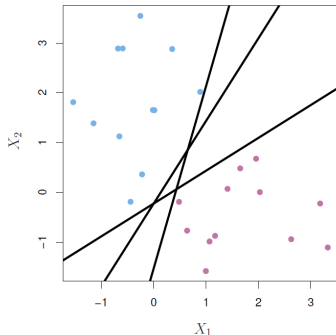
- If $X = (X_1, \dots, X_p)^T$ satisfies (1), then X **lies on the hyperplane**.
- If $X = (X_1, \dots, X_p)^T$ does not satisfy (1), then X lies to **one side** of the hyperplane.
- Which side of the hyperplane a point lies can be determined by calculating the **sign** of the L.H.S of (1).
- Note that a hyperplane is a **line** when $p = 2$.

Hyperplane in Two-dimensional Space



- The hyperplane $1 + 2X_1 + 3X_2 = 0$ is shown
 - Blue region : $1 + 2X_1 + 3X_2 > 0$
 - Purple region : $1 + 2X_1 + 3X_2 < 0$

Separating Hyperplanes



- If $f(X) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$, then $f(X) > 0$ for points on one side of the hyperplane, and $f(X) < 0$ for points on the other.
- If we code the colored points as $Y_i = 1$ for blue, and $Y_i = -1$ for pink, then if $Y_i \cdot f(X_i) > 0$ for all i , $f(X_i) = 0$ defines a separating hyperplane.

Separating Hyperplanes

- If a **separating hyperplane** exists, we can use it to construct a very natural classifier.
- For a test observation x^* ,

$$f(x^*) = \beta_0 + \beta_1 x_1^* + \cdots + \beta_p x_p^*$$

- If $f(x^*) > 0$, we assign the test observation to class 1
- If $f(x^*) < 0$, we assign the test observation to class 2
- For the **magnitude** of $f(x^*)$,
 - If $|f(x^*)|$ is relatively large, the class assignment is **confident**.
 - If $|f(x^*)|$ is relatively small, the class assignment is **less confident**.

Maximal Margin Classifier

- The **maximal margin hyperplane** is the separating hyperplane that is farthest from the training observations.
 - Among all separating hyperplanes, find the one that makes the biggest gap or **margin** between the two classes.
- Constrained optimization problem:

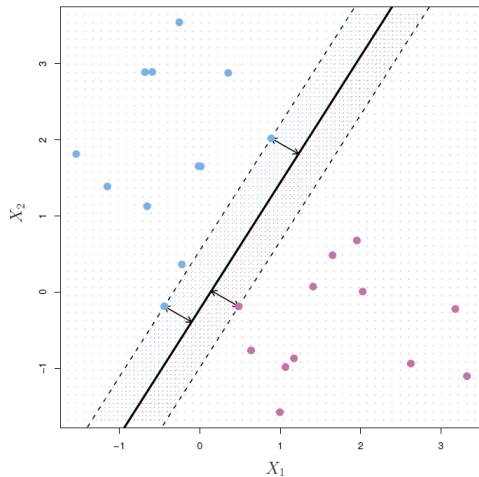
maximize M subject to
 $\beta_0, \beta_1, \dots, \beta_p$

$$\sum_{j=1}^p \beta_j^2 = 1, \text{ and } y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M$$

for all $i = 1, 2, \dots, n$.

- This can be rephrased as a convex quadratic program, and solved efficiently. The function **svm()** in an R package “**e1071**” solves this problem efficiently.

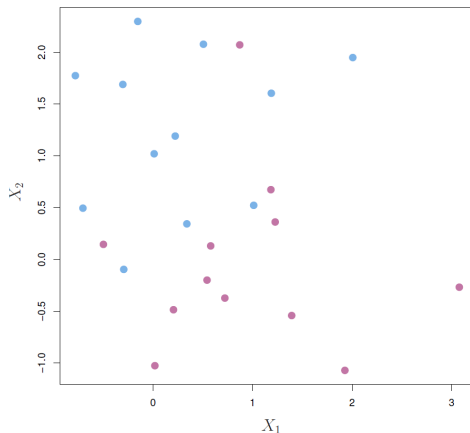
Maximal Margin Classifier



The Non-separable Case

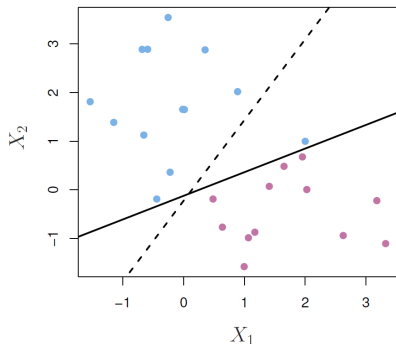
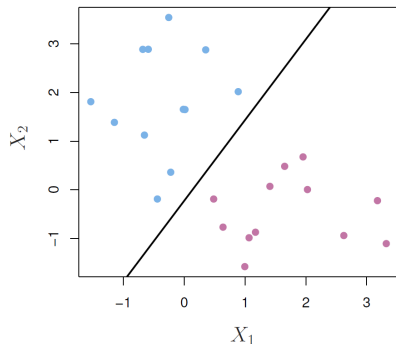
- The **maximal margin classifier** is a very natural way to perform classification, if a separating hyperplane exists.
- However, in many cases **no** separating hyperplane exists, and so there is **no** maximal margin classifier.
- If a separating hyperplane doesn't exist, there is **no solution** to M in the optimization problem.
- However, we can extend the concept of a separating hyperplane in order to develop a hyperplane that **almost** separates the classes, using a so-called **soft margin**.
- The generalization of the maximal margin classifier to the non-separable case is known as the **support vector classifier**.

Non-separable Data



- The data above are **not separable** by a linear boundary.
- This is often the case, unless $n < p$.

Separable but Noisy Data

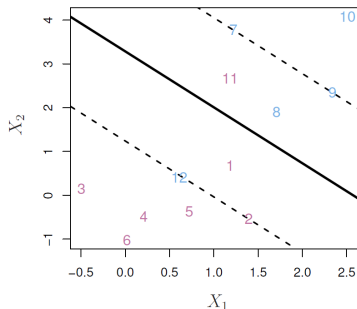
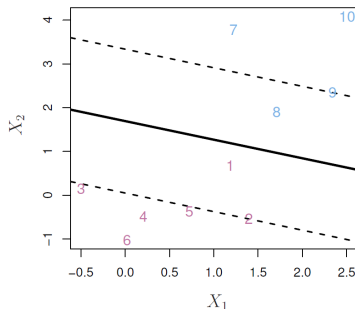


- Sometimes the data are separable, but noisy. This can lead to a poor solution for the maximal-margin classifier.
- The support vector classifier maximizes a soft margin.

Support Vector Classifier

- We need to consider a classifier based on a hyperplane that does **not perfectly separate** the two classes, in the interest of
 - Greater robustness to individual observations
 - Better classification of most of the training observations
- That is, it could be worthwhile to **misclassify a few** training observations in order to do a better job in classifying the remaining observations.
- The **support vector classifier** (**soft margin classifier**) allows some observations to be on the **incorrect** side of the margin, or even the **incorrect** side of the hyperplane.
- The margin is **soft** because it can be violated by some of the training observations.
- The **support vector classifier** classifies a test observation depending on which side of a hyperplane it lies.

Optimization of Support Vector Classifier

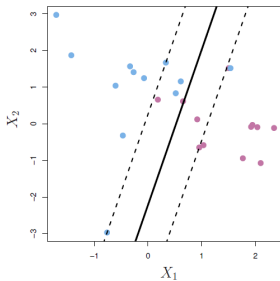
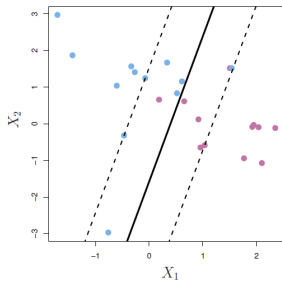
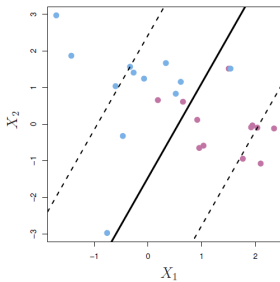
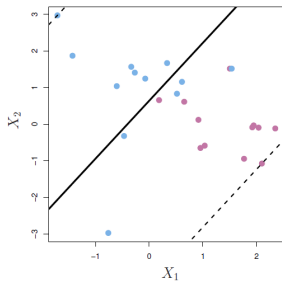


$$\underset{\beta_0, \beta_1, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n}{\text{maximize}} \quad M \text{ subject to } \sum_{j=1}^p \beta_j^2 = 1,$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i),$$

$$\epsilon_i \geq 0, \text{ and } \sum_{i=1}^n \epsilon_i \leq C, \text{ where } C > 0 \text{ is a tuning parameter.}$$

Support Vector Classifier



Margins and Slack Variables

- M is the width of the **margin** — as large as possible.
- $\epsilon_1, \dots, \epsilon_n$ are **slack variables**.
 - If $\epsilon_i = 0$, the i th obs. is on the **correct** side of the **margin**.
 - If $\epsilon_i > 0$, the i th obs. is on the **wrong** side of the **margin**.
 - If $\epsilon_i > 1$, the i th obs. is on the **wrong** side of the **hyperplane**.
- A **regularization** parameter C bounds the sum of the ϵ_i 's
 - If C is small, we seek narrow **margins** that are rarely violated — low bias but high variance.
 - If C is larger, the **margin** is wider and we allow more violations to it — more bias but lower variance.
- Observations that lie directly on the margin, or on the wrong side of the margin for their class are known as **support vectors**.
- These observations do affect the **support vector classifier**.


```
## Simple example (simulate data set)
set.seed(1)
x <- matrix(rnorm(20*2), ncol=2)
y <- c(rep(-1, 10), rep(1, 10))
x[y==1, ] <- x[y==1, ] + 1
plot(x, col=(3-y), pch=19, xlab="X1", ylab="X2")
```

```
## Support vector classifier with cost=10
library(e1071)
dat <- data.frame(x, y=as.factor(y))
svmfit <- svm(y~., data=dat, kernel="linear", cost=10,
              scale=FALSE)
plot(svmfit, dat)
summary(svmfit)
```

```
names(svmfit)
svmfit$index
svmfit$coefs
```

```
## coefficient beta_0 (negative)
beta0 <- svmfit$rho
```

```
## coefficient beta_1 and beta_2
beta <- drop(t(svmfit$coefs)%*%x[svmfit$index,])
```

```
plot(x, col=(3-y), pch=19, xlab="X1", ylab="X2")
points(x[svmfit$index, ], pch=5, cex=2)
abline(beta0 / beta[2], -beta[1] / beta[2])
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)
```

```
## Different value of tuning parameter (cost=0.1)
svmfit <- svm(y~., data=dat, kernel="linear", cost=0.1,
              scale=FALSE)

svmfit$index
beta <- drop(t(svmfit$coefs)%*%x[svmfit$index,])
beta0 <- svmfit$rho
```

```
plot(x, col=(3-y), pch=19, xlab="X1", ylab="X2")
points(x[svmfit$index, ], pch=5, cex=2)
abline(beta0 / beta[2], -beta[1] / beta[2])
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)
```

```
## A function to create a grid of values or a lattice of values
make.grid <- function(x, n = 75) {
  ran <- apply(x, 2, range)
  x1 <- seq(from = ran[1,1], to = ran[2,1], length = n)
  x2 <- seq(from = ran[1,2], to = ran[2,2], length = n)
  expand.grid(X1 = x1, X2 = x2)
}
```

```
xgrid <- make.grid(x)
xgrid[1:76,]
```

```
## Classification of potential test set
ygrid <- predict(svmfit, xgrid)
plot(xgrid, col=c("red", "blue")[as.numeric(ygrid)], pch=20,
     cex=.2)
points(x, col=y+3, pch=19)
points(x[svmfit$index, ], pch=5, cex=2)
```

```
## Cross validation to find the optimal tuning parameter (cost)
set.seed(1)
tune.out <- tune(svm, y~., data=dat, kernel="linear",
                 ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune.out)
bestmod <- tune.out$best.model
summary(bestmod)
```

```
## Generate test set
set.seed(4321)
xtest <- matrix(rnorm(20*2), ncol=2)
ytest <- sample(c(-1,1), 20, rep=TRUE)
xtest[ytest==1, ] <- xtest[ytest==1, ] + 1
testdat <- data.frame(xtest, y=as.factor(ytest))
```

```
## Compute misclassification rate for the optimal model
ypred <- predict(bestmod, testdat)
table(predict=ypred, truth=testdat$y)
mean(ypred!=testdat$y)
```

```
## Misclassification rate for other value of cost (cost=0.01)
svmfit <- svm(y~., data=dat, kernel="linear", cost=.01,
              scale=FALSE)
ypred <- predict(svmfit, testdat)
table(predict=ypred, truth=testdat$y)
mean(ypred!=testdat$y)
```

```
## Misclassification rate for other value of cost (cost=1)
svmfit <- svm(y~., data=dat, kernel="linear", cost=1,
              scale=FALSE)
ypred <- predict(svmfit, testdat)
table(predict=ypred, truth=testdat$y)
mean(ypred!=testdat$y)
```

```
## Misclassification rate for other value of cost (cost=100)
svmfit <- svm(y~., data=dat, kernel="linear", cost=100,
              scale=FALSE)
ypred <- predict(svmfit, testdat)
table(predict=ypred, truth=testdat$y)
mean(ypred!=testdat$y)
```

```
library(mnormt)
library(e1071)
```

```
## Misclassification Rate of SVC
SVC.MCR <- function(x.tran, x.test, y.tran, y.test,
                    cost=c(0.01,0.1,1,10,100)) {
  dat <- data.frame(x.tran, y=as.factor(y.tran))
  testdat <- data.frame(x.test, y=as.factor(y.test))
  MCR <- rep(0, length(cost)+1)
  for (i in 1:length(cost)) {
    svmfit <- svm(y~., data=dat, kernel="linear",
                  cost=cost[i])
    MCR[i] <- mean(predict(svmfit, testdat)!=testdat$y)
  }
  tune.out <- tune(svm, y~., data=dat, kernel="linear",
                  ranges=list(cost=cost))
  pred <- predict(tune.out$best.model, testdat)
  MCR[length(cost)+1] <- mean(pred!=testdat$y)
MCR
}
```

```
set.seed(123)
K <- 100
RES <- matrix(NA, K, 6)
colnames(RES) <- c("0.01", "0.1" , "1" , "10" ,"100", "CV")
```

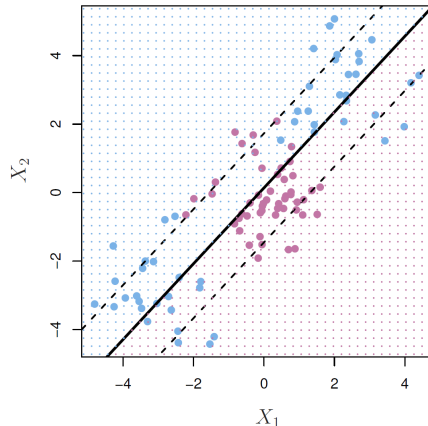
```
for (i in 1:K) {
  x.A <- rmnorm(100, rep(0, 2), matrix(c(1,-0.5,-0.5,1),2))
  x.B <- rmnorm(100, rep(1, 2), matrix(c(1,-0.5,-0.5,1),2))
  x.tran <- rbind(x.A[1:50, ], x.B[1:50, ])
  x.test <- rbind(x.A[-c(1:50), ], x.B[-c(1:50), ])
  y.tran <- factor(rep(0:1, each=50))
  y.test <- factor(rep(0:1, each=50))
  RES[i,] <- SVC.MCR(x.tran, x.test, y.tran, y.test)
}
```

```
apply(RES, 2, summary)
boxplot(RES, boxwex=0.5, col=2:7,
        names=c("0.01", "0.1", "1", "10", "100", "CV"),
        main="", ylab="Classification Error Rates")
```

Non-linear Decision Boundaries

- The support vector classifier is a natural approach for classification in the two-class setting, if the boundary between the two classes is **linear**.
- In practice, we can be faced with **non-linear** class boundaries.
- The performance of **linear** regression can suffer when there is a **nonlinear** relationship between the predictors and the outcome.
- We need to consider enlarging the feature space using functions of the predictors such as **quadratic** and **cubic** terms, in order to address this **non-linearity**.
- In the case of the support vector classifier, we can also **enlarge the feature space** using quadratic, cubic, and even higher-order polynomial functions of the predictors.

Failure of Linear Boundary



- Sometime a linear boundary simply won't work, no matter what value of C .
- The example above is such a case. What to do?

Feature Expansion

- Enlarge the space of features by including transformations;

$$X_1^2, X_1^3, X_1X_2, X_1X_2^2, \dots$$

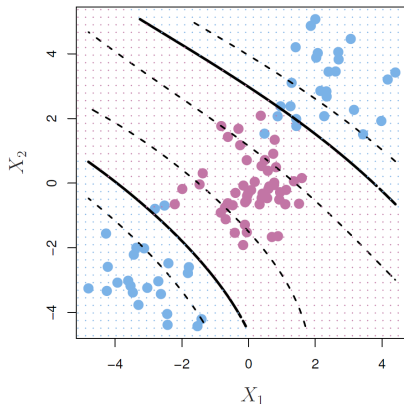
Hence, go from a p -dimensional space to a M -dimensional space ($M > p$).

- Fit a support-vector classifier in the enlarged space.
- This results in non-linear decision boundaries in the space.
- For example,

$$\beta_0 + \beta_1X_1 + \beta_2X_2 + \beta_3X_1^2 + \beta_4X_2^2 + \beta_5X_1X_2 = 0$$

This leads to nonlinear decision boundaries in the original space (quadratic conic sections).

Cubic Polynomials



- Here we use a basis expansion of **cubic polynomials**.
- The support vector classifier in the enlarged space solves the problem in the lower-dimensional space.

Computational Issues for High-dimensional Polynomials

- Cubic polynomial is

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1^2 + \beta_4 X_2^2 + \beta_5 X_1 X_2 + \beta_6 X_1^3 + \beta_7 X_2^3 + \beta_8 X_1 X_2^2 + \beta_9 X_1^2 X_2 = 0$$

- However, high-dimensional **polynomials** get wild rather fast.
- There are many possible ways to enlarge the feature space, but computations would become **unmanageable** for a huge number of features .
- There is a more elegant and controlled way to introduce nonlinearities in support vector classifiers, using **kernels**.
- Before we discuss these, we must understand the role of **inner products** in support vector classifiers.

Inner Products

- Inner product between vectors

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

- The linear support vector classifier can be represented as

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

- To estimate the parameters $\alpha_1, \dots, \alpha_n$ and β_0 , all we need are the $\binom{n}{2}$ inner products $\langle x_i, x_{i'} \rangle$ between all pairs of training observations.
- To evaluate $f(x)$, we need to compute the inner product between the new point x and each of the training points x_i .

Kernels and Support Vector Machines

- It turns out that most of the $\hat{\alpha}_i$ can be **zero**:

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

\mathcal{S} is the **support set** of indices i such that $\hat{\alpha}_i > 0$.

- A generalization of the inner product of the form $K(x_i, x_{i'})$ is called a **kernel**.
- A **kernel** is a function that quantifies the similarity of two observations.
- When the support vector classifier is combined with a non-linear kernel, the resulting classifier is known as a **support vector machine**.

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

Examples of Kernels

- Standard linear kernel

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

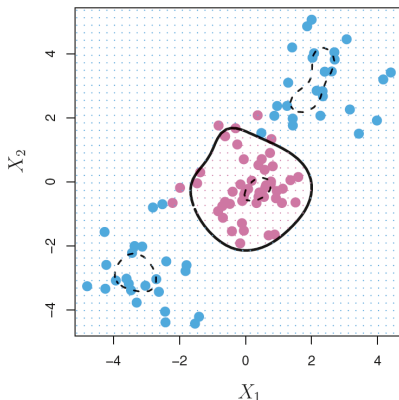
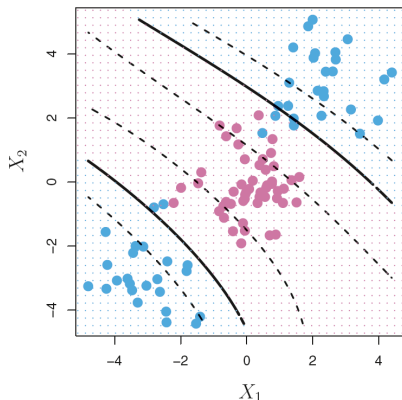
- Polynomial kernel of degree $d > 1$

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

- Radial kernel

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

Polynomial Kernel and Radial Kernel



- Left: SVM with a polynomial kernel of degree 3
- Right: SVM with a radial kernel

Computational Advantages

- One advantage of using kernels over enlarging the feature space using functions of the original features is **computational efficiency**.
 - Just compute $K(x_i, x'_i)$ for all $\binom{n}{2}$ distinct pairs i and i' .
 - This can be done without explicitly working in the enlarged feature space.
- This is important because in many applications of SVMs, the enlarged feature space is so large that computations are **intractable**.
- For **radial kernel**, the feature space is implicit and infinite-dimensional, so we could never do the computations there anyway.

```
## Simulate non-linear data set
set.seed(1)
x <- matrix(rnorm(200*2), ncol=2)
x[1:100, ] <- x[1:100, ] + 2
x[101:150, ] <- x[101:150, ] - 2
y <- c(rep(-1, 150), rep(1, 50))
dat <- data.frame(x, y=as.factor(y))
plot(x, col=y+3, pch=19)
```

```
fit <- svm(y~.,data=dat, kernel="radial", gamma=0.5, cost=0.1)
plot(fit, dat)
summary(fit)
```

```
fit <- svm(y~.,data=dat, kernel="radial", gamma=0.5, cost=5)
plot(fit, dat)
summary(fit)
```

```
fit <- svm(y~.,data=dat, kernel="radial", gamma=0.5, cost=1)
```

```
px1 <- seq(round(min(x[,1]),1), round(max(x[,1]),1), 0.1)
px2 <- seq(round(min(x[,2]),1), round(max(x[,2]),1), 0.1)
xgrid <- expand.grid(X1=px1, X2=px2)
ygrid <- as.numeric(predict(fit, xgrid))
ygrid[ygrid==1] <- -1
ygrid[ygrid==2] <- 1
```

```
plot(xgrid, col=ygrid+3, pch = 20, cex = .2)
points(x, col = y+3, pch = 19)
```

```
pred <- predict(fit, xgrid, decision.values=TRUE)
func <- attributes(pred)$decision
contour(px1, px2, matrix(func, length(px1), length(px2)),
        level=0, col="purple", lwd=2, lty=2, add=TRUE)
```

```
## Separate training and test sets
set.seed(1234)
tran <- sample(200, 100)
test <- setdiff(1:200, tran)
```

```
gamma <- c(0.5, 1, 5, 10)
cost <- c(0.01, 1, 10, 100)
R <- NULL
```

```
for (i in 1:length(gamma)) {
  for (j in 1:length(cost)) {
    svmfit <- svm(y~., data=dat[tran, ], kernel="radial",
                  gamma=gamma[i] , cost=cost[j])
    pred <- predict(svmfit, dat[test, ])
    R0 <- c(gamma[i], cost[j], mean(pred!=dat[test, "y"]))
    R <- rbind(R, R0)
  }
}
```

```
colnames(R) <- c("gamma", "cost", "error")
rownames(R) <- seq(dim(R)[1])
R
```

```
## Find the optimal tuning parameters
set.seed(1)
tune.out <- tune(svm, y~., data=dat[tran, ], kernel="radial",
               ranges=list(gamma=gamma, cost=cost))
summary(tune.out)
tune.out$best.parameters
```

```
pred <- predict(tune.out$best.model, dat[test,])
table(pred=pred, true=dat[test, "y"])
mean(pred!=dat[test, "y"])
```

```
degree <- c(1, 2, 3, 4)
R <- NULL
```

```

for (i in 1:length(degree)) {
  for (j in 1:length(cost)) {
    svmfit <- svm(y~., data=dat[tran, ], kernel="polynomial",
                  degree=degree[i] , cost=cost[j])
    pred <- predict(svmfit, dat[test, ])
    R0 <- c(degree[i], cost[j], mean(pred!=dat[test, "y"]))
    R <- rbind(R, R0)
  }
}

```

```

colnames(R) <- c("degree", "cost", "error")
rownames(R) <- seq(dim(R)[1])
R

```

```

tune.out <- tune(svm, y~., data=dat[tran, ], kernel="polynomial",
                 ranges=list(degree=degree, cost=cost))
summary(tune.out)
tune.out$best.parameters

```

```
pred <- predict(tune.out$best.model, dat[test,])
table(pred=pred, true=dat[test, "y"])
mean(pred!=dat[test, "y"])
```

```
R <- NULL
for (i in 1:length(gamma)) {
  for (j in 1:length(cost)) {
    svmfit <- svm(y~., data=dat[tran, ], kernel="sigmoid",
                  gamma=gamma[i] , cost=cost[j])
    pred <- predict(svmfit, dat[test, ])
    R0 <- c(gamma[i], cost[j], mean(pred!=dat[test, "y"]))
    R <- rbind(R, R0)
  }
}
```

```
colnames(R) <- c("gamma", "cost", "error")
rownames(R) <- seq(dim(R)[1])
R
```

```
tune.out <- tune(svm, y~., data=dat[tran, ], kernel="sigmoid",  
                ranges=list(gamma=gamma, cost=cost))  
summary(tune.out)  
tune.out$best.parameters
```

```
pred <- predict(tune.out$best.model, dat[test,])  
table(pred=pred, true=dat[test, "y"])  
mean(pred!=dat[test, "y"])
```

```
set.seed(123)  
N <- 20  
RES <- matrix(0, N, 3)  
colnames(RES) <- c("radial", "poly", "sigmoid")
```



```
for (i in 1:N) {  
  tran <- sample(200, 100)  
  test <- setdiff(1:200, tran)  
  
  tune1 <- tune(svm, y~., data=dat[tran, ], kernel="radial",  
               ranges=list(gamma=gamma, cost=cost))  
  pred1 <- predict(tune1$best.model, dat[test,])  
  RES[i, 1] <- mean(pred1!=dat[test, "y"])  
  
  tune2 <- tune(svm, y~., data=dat[tran, ], kernel="polynomial",  
               ranges=list(degree=degree, cost=cost))  
  pred2 <- predict(tune2$best.model, dat[test,])  
  RES[i, 2] <- mean(pred2!=dat[test, "y"])  
  
  tune3 <- tune(svm, y~., data=dat[tran, ], kernel="sigmoid",  
               ranges=list(gamma=gamma, cost=cost))  
  pred3 <- predict(tune3$best.model, dat[test,])  
  RES[i, 3] <- mean(pred3!=dat[test, "y"])  
}
```

```
apply(RES, 2, summary)
```

Example: Heart Data

- These data contain a binary outcome **AHD** for 303 patients who presented with chest pain.
- An outcome value of **Yes** indicates the presence of heart disease based on an angiographic test, while **No** means no heart disease.
- There are 13 predictors including **Age**, **Sex**, **Chol** (a cholesterol measurement), and other heart and lung function measurements.
- We can find some **missing values** from data, so we need to remove the samples with **missing values**. Then, the sample size is reduced down to 297.

```
url.ht <- "https://www.statlearning.com/s/Heart.csv"
Heart <- read.csv(url.ht, h=T)
summary(Heart)
```

```
Heart <- Heart[, colnames(Heart)!="X"]
Heart[, "Sex"] <- factor(Heart[, "Sex"], 0:1, c("female", "male"))
Heart[, "Fbs"] <- factor(Heart[, "Fbs"], 0:1, c("false", "true"))
Heart[, "ExAng"] <- factor(Heart[, "ExAng"], 0:1, c("no", "yes"))
Heart[, "ChestPain"] <- as.factor(Heart[, "ChestPain"])
Heart[, "Thal"] <- as.factor(Heart[, "Thal"])
Heart[, "AHD"] <- as.factor(Heart[, "AHD"])
```

```
summary(Heart)
dim(Heart)
sum(is.na(Heart))
```

```
Heart <- na.omit(Heart)
dim(Heart)
summary(Heart)
```

```
## Separate training and test sets
set.seed(123)
train <- sample(1:nrow(Heart), nrow(Heart)/2)
test <- setdiff(1:nrow(Heart), train)
```

```
## Classification tree
library(tree)
tree.tran <- tree(AHD ~., Heart, subset = train)
tree.pred <- predict(tree.tran, Heart[test,], type="class")
mean(tree.pred!=Heart$AHD[test])
```

```
## Cross-validated tree
set.seed(1234)
cv.heart <- cv.tree(tree.tran, FUN=prune.misclass, K=5)
w <- cv.heart$size[which.min(cv.heart$dev)]
prune.heart <- prune.misclass(tree.tran, best=w)
heart.pred <- predict(prune.heart, Heart[test,], type="class")
mean(heart.pred!=Heart$AHD[test])
```

```
set.seed(1111)
```

```
## Bagging and random forest (m=2, 4, and 6)
library(randomForest)
bag.heart <- randomForest(x=Heart[train,-14], y=Heart[train,14],
                          xtest=Heart[test,-14], ytest=Heart[test,14], mtry=13)
bag.heart$test$err.rate[500,1]
```

```
rf.heart <- randomForest(x=Heart[train,-14], y=Heart[train,14],
                        xtest=Heart[test,-14], ytest=Heart[test,14], mtry=6)
rf.heart$test$err.rate[500,1]
```

```
rf.heart <- randomForest(x=Heart[train,-14], y=Heart[train,14],
                        xtest=Heart[test,-14], ytest=Heart[test,14], mtry=4)
rf.heart$test$err.rate[500,1]
```

```
rf.heart <- randomForest(x=Heart[train,-14], y=Heart[train,14],
                        xtest=Heart[test,-14], ytest=Heart[test,14], mtry=2)
rf.heart$test$err.rate[500,1]
```

```
library(e1071)
```

```
## SVM with a linear kernel
tune.out <- tune(svm, AHD~., data=Heart[train, ],
                kernel="linear", ranges=list(
                  cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
heart.pred <- predict(tune.out$best.model, Heart[test,])
table(heart.pred, Heart$AHD[test])
mean(heart.pred!=Heart$AHD[test])
```

```
## SVM with a radial kernel
tune.out <- tune(svm, AHD~., data=Heart[train, ],
                kernel="radial", ranges=list(
                  cost=c(0.1,1,10,100), gamma=c(0.5,1,2,3)))
heart.pred <- predict(tune.out$best.model, Heart[test,])
table(heart.pred, Heart$AHD[test])
mean(heart.pred!=Heart$AHD[test])
```

```
## SVM with a polynomial kernel
tune.out <- tune(svm, AHD~.,data=Heart[train, ],
               kernel="polynomial", ranges=list(
                 cost=c(0.1,1,10,100), degree=c(1,2,3)))
heart.pred <- predict(tune.out$best.model, Heart[test,])
table(heart.pred, Heart$AHD[test])
mean(heart.pred!=Heart$AHD[test])
```

```
## SVM with a sigmoid kernel
tune.out <- tune(svm, AHD~.,data=Heart[train, ],
               kernel="sigmoid", ranges=list(
                 cost=c(0.1,1,10,100), gamma=c(0.5,1,2,3)))
heart.pred <- predict(tune.out$best.model, Heart[test,])
table(heart.pred, Heart$AHD[test])
mean(heart.pred!=Heart$AHD[test])
```

```
set.seed(123)
N <- 20
Err <- matrix(0, N, 5)
```

```

for (i in 1:N) {
  train <- sample(1:nrow(Heart), floor(nrow(Heart)*2/3))
  test <- setdiff(1:nrow(Heart), train)

  g1 <- randomForest(x=Heart[train,-14], y=Heart[train,14],
                     xtest=Heart[test,-14], ytest=Heart[test,14], mtry=4)
  Err[i,1] <- g1$test$err.rate[500,1]

  g2 <- tune(svm, AHD~., data=Heart[train, ], kernel="linear",
             ranges=list(cost=c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
  p2 <- predict(g2$best.model, Heart[test,])
  Err[i,2] <- mean(p2!=Heart$AHD[test])

  g3 <- tune(svm, AHD~., data=Heart[train, ], kernel="radial",
             ranges=list(cost=c(0.1,1,10,100), gamma=c(0.5,1,2,3)))
  p3 <- predict(g3$best.model, Heart[test,])
  Err[i,3] <- mean(p3!=Heart$AHD[test])
}

```



```

g4 <- tune(svm, AHD~.,data=Heart[train, ],kernel="polynomial",
           ranges=list(cost=c(0.1,1,10,100), degree=c(1,2,3)))
p4 <- predict(g4$best.model, Heart[test,])
Err[i,4] <- mean(p4!=Heart$AHD[test])

g5 <- tune(svm, AHD~.,data=Heart[train, ],kernel="sigmoid",
           ranges=list(cost=c(0.1,1,10,100), gamma=c(0.5,1,2,3)))
p5 <- predict(g5$best.model, Heart[test,])
Err[i,5] <- mean(p5!=Heart$AHD[test])
}

```

```

labels <- c("RF","SVM.linear","SVM.radial","SVM.poly","SVM.sig")
boxplot(Err, boxwex=0.5, main="Random Forest and SVM", col=2:6,
        names=labels, ylab="Classification Error Rates",
        ylim=c(0,0.4))

```

```

colnames(Err) <- labels
apply(Err, 2, summary)

```