# 11 Support vector machines

**Support vector machines** (SVMs) are mostly used for classification of binary outcomes. There is a regression version of SVM, which is like regularized regression. We only focus on **classification SVMs**. For ease of mathematical presentation, we suppose the two outcome categories are $\{-1, 1\}$. In practice, we often need to code the outcome as a factor. (In `svm()`, a numerical outcome variable triggers regression SVM unless `type=` is set.)

## 11.1 Perfectly separable data (ISLR 9.1)

A **hyperplane** in a $p$-dimensional space is a $(p-1)$-dimensional linear subspace. It can always be represented as

$$\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0. \tag{9.2}$$

Viewed from the perspective of projection: Let $\beta = (\beta_1, \ldots, \beta_p)$ and $x = (x_1, \ldots, x_p)$. Then (9.2) can be written as

$$\beta_0 + \langle \beta, x \rangle = 0, \text{ or equivalently, } \frac{1}{\|\beta\|_2} \langle \beta, x \rangle = -\frac{\beta_0}{\|\beta\|_2}.$$

where $\|\beta\|_2$ is the length of vector $\beta$. The left hand side, $\frac{1}{\|\beta\|_2}\langle \beta, x \rangle$, is the **projected value** when $x$ is projected onto $\beta$; the left hand side, $-\frac{\beta_0}{\|\beta\|_2}$, is a constant given $\beta$ and $\beta_0$. Thus,

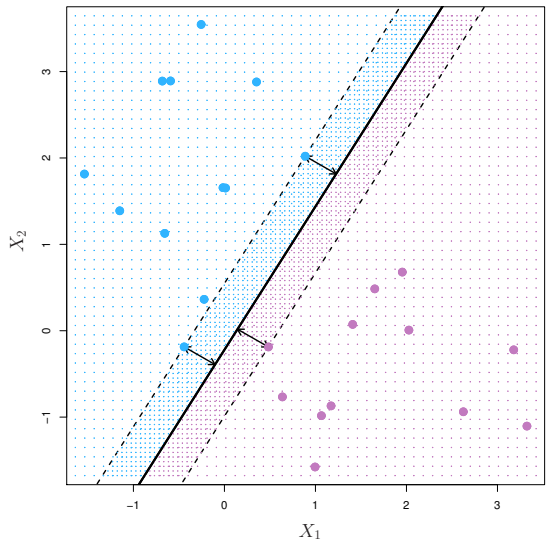
- The hyperplane (9.2) consists of those $x$ that have the same projected value $-\frac{\beta_0}{\|\beta\|_2}$ when projected onto $\beta$.
- The hyperplane is **perpendicular to** $\beta$.
- We can simplify the notation by only considering those $\beta$ with $\|\beta\|_2 = 1$: then $\langle \beta, x \rangle = -\beta_0$.

**Perfectly separating hyperplanes**: To motivate SVM, we first consider binary data that can be *perfectly separated* in the feature space. In this situation, there are many separating hyperplanes. For any separating hyperplane $\beta_0 + \langle \beta, x \rangle = 0$, we have a classifier:

- $\beta_0 + \langle \beta, x_i \rangle > 0$ for all observations with $y_i = 1$;
- $\beta_0 + \langle \beta, x_i \rangle < 0$ for all observations with $y_i = -1$.

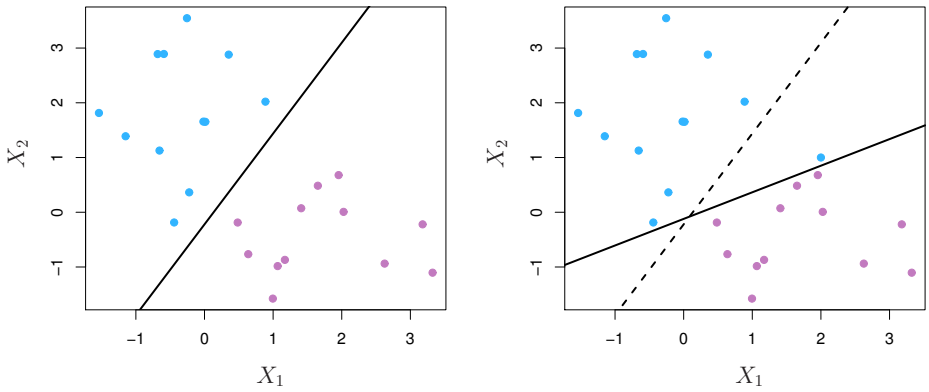
If it is the other way around we can always flip the signs of $\beta$ and $\beta_0$ so that the above will hold. A separating hyperplane is at the center of its corresponding **margin**, which is a separating region with a constant width.

The **optimal separating hyperplane** (or **maximal margin hyperplane**) is the one with the largest margin (i.e., the largest minimum distance from all observations to the separating hyperplane). The corresponding classifier is called the *maximal margin classifier* (Figure 9.3). This is "hard margin classification".




The **support vectors** are those observations that define the margin. The optimal hyperplane $\hat{\beta}_0 + \langle \hat{\beta}, x \rangle = 0$ is also called the **decision boundary**, and $\hat{\beta}_0 + \langle \hat{\beta}, x_i \rangle$ is the **decision value** for observation $i$. Notes:

- The optimal separating hyperplane depends only on a few observations. It can be very sensitive (Figure 9.5).
- This will not work when data cannot be perfectly separated in the feature space.
- "Soft margin classification" (in next section) can help address both issues.




**Mathematical definition**: Given a separating hyperplane $\beta_0 + \langle \beta, x_i \rangle = 0$ with $\|\beta\|_2 = 1$. Let $f_i = \beta_0 + \langle \beta, x_i \rangle$. The smallest distance to the hyperplane among those with $y_i = 1$ is $\min_{y_i=1} y_i f_i$, and that among those with $y_i = -1$ is $\min_{y_i=-1} y_i f_i$. So the margin of the hyperplane is $\min y_i f_i$. We now have an optimization problem:

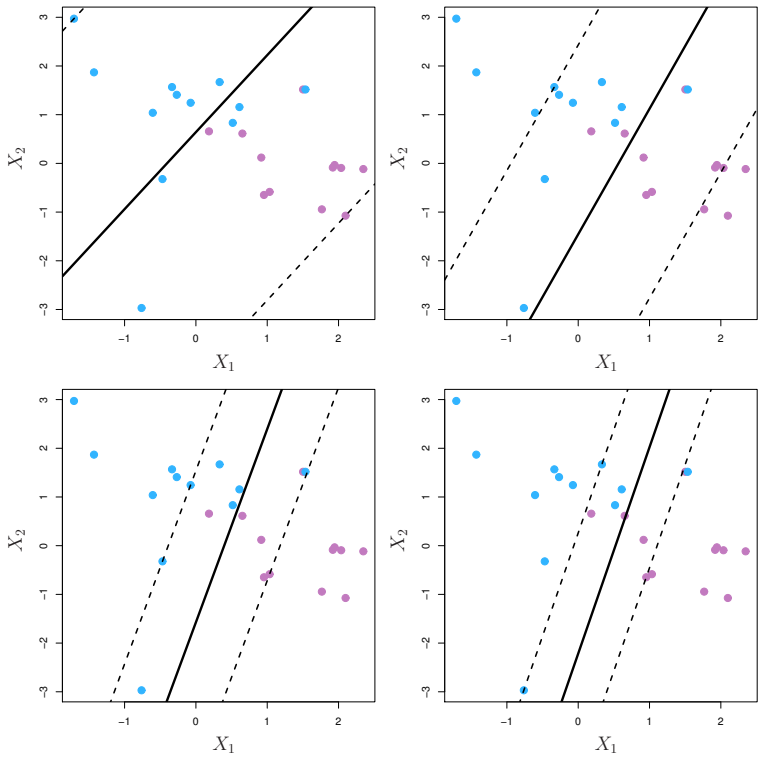
$$\max_{\beta_0,\beta} M \quad \text{subject to } \|\beta\|_2 = 1,\ y_i f_i \geq M \quad (i = 1, \cdots, n). \tag{9.9–9.11}$$

An alternative expression is:

$$\min \|\beta\|_2 \quad \text{subject to } y_i f_i \geq 1 \quad (i = 1, \cdots, n).$$

## 11.2 Linear SVMs (ISLR 9.2)

When the data are not perfectly separable in the feature space, we may allow some observations to be on the "wrong" side of the margin. We give "allowances" to the observations but control the total "budget", $B$. Illustration in Figure 9.7. With a large $B$ (top left), the margin is large and many observations can be support vectors. With a small $B$ (bottom right), the margin is small and only a few observations can be support vectors.



The formal definition is below. The observations that define the optimal hyperplane and margin are called **support vectors**. It can be shown that the SVM solution $\hat{\beta}$ satisfies

$$\hat{\beta} = \sum_i \alpha_i x_i, \text{ with } \sum_i \alpha_i = 0. \tag{1}$$

Those with $\alpha_i \neq 0$ are the support vectors (because they define $\hat{\beta}$).

Notes:

- **Scale the input variables** unless you have a reason not to. Otherwise the results can be an artifact of the choice of the units for the input variables. Scaling is the default in `svm()` (`scale=T`).
- $B$ is a hyperparameter. (ISLR uses notation $C$, which could be confused with "cost".)
- In `svm()`, `cost` is used instead of $B$; `cost` has an opposite effect of $B$.
- A large "budget" (or a small "cost") tends to lead to a wider margin and more support vectors.

**Mathematical definition**: Let $f_i = \beta_0 + \langle \beta, x_i \rangle$. The most intuitive definition is to solve the optimization problem:

$$\max_{\beta_0,\beta,\epsilon} M \quad \text{subject to } \|\beta\|_2 = 1,\ y_i f_i \geq M(1-\epsilon_i),\ \epsilon_i \geq 0,\ \sum_i \epsilon_i \leq B \quad (i = 1, \cdots, n). \tag{9.12–9.15}$$

- Here, $M$ is the distance between the hyperplane and margin boundaries.
- The "allowances" $\epsilon_i$'s are called **slack variables**.
- Observations with $\epsilon_i > 0$ are the **support vectors**. They are on the "wrong" side of the margin.
- Observations with $\epsilon_i > 1$ are on the "wrong" side of the hyperplane.

An alternative expression is:

$$\min \|\beta\|_2 \quad \text{subject to } \sum_i (1 - y_i f_i)_+ \leq B.$$

This is similar to the alternative expression of ridge regresion. In fact, this can be alternatively expressed as

$$\min_{\beta_0,\beta} \frac{1}{2}\|\beta\|_2^2 + C \sum_i (1 - y_i f_i)_+, \tag{2}$$

where $C \geq 0$ is the **cost**, or as

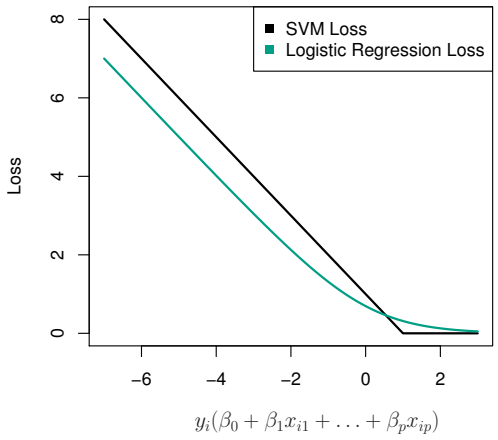
$$\min_{\beta_0,\beta} \sum_i (1 - y_i f_i)_+ + \lambda \|\beta\|_2^2, \tag{3}$$

where $\lambda = \frac{1}{2C} \geq 0$. The function $L(y_i, f_i) = (1 - y_i f_i)_+$ is called the **hinge loss**. **Linear SVMs are ridged regression with the hinge loss**. All support vectors have non-zero hinge loss. That is, the support vectors in class $y_i = 1$ have $f_i < 1$, and the support vectors in class $y_i = -1$ have $f_i > -1$.

If we replace the hinge loss by the **binomial loss** $L(y_i, f_i) = \log(1 + e^{-y_i f_i})$, we have the **ridged logistic regression**,

$$\min_{\beta_0,\beta} \sum_i \log[1 + e^{-y_i(\beta_0 + x_i'\beta)}] + \lambda \|\beta\|_2^2. \tag{4}$$

Note that $\sum_i \log[1 + e^{-y_i(\beta_0 + x_i'\beta)}]$ is the negative log-likelihood for logistic regression. The hinge and binomial loss functions are similar (Figure 9.12). Thus the linear SVM (3) and the ridged logistic regression (4) are similar.



## 11.3 The R e1071 package

When using `svm()` in the R e1071 package for classification, make sure the outcome is coded as a factor. Otherwise a regression SVM would be performed. The first category of the factor is treated as the target category (as if it is $y = 1$) and the SVM model is fit so that the first category is associated with positive decision values.

We use the `Heart` dataset as an example. First, we prepare the data.

```
Heart = read.csv("Heart.csv", row.names=1)  ## first column is row name, not a variable
names(Heart); dim(Heart)  ## 303, 14
Heart$Thal = factor(Heart$Thal, c('normal', 'reversable', 'fixed'), ordered=T)
Heart$ChestPain = factor(Heart$ChestPain, levels(Heart$ChestPain), ordered=T)
table(Heart$AHD)  ## 164 No, 139 Yes

set.seed(2018)
trainidx = sample(1:nrow(Heart), 220)
Heart.train = Heart[trainidx, ]  ## training set
Heart.test = Heart[-trainidx, ]  ## test set

colahd = (Heart.train$AHD == 'Yes') + 1  ## 'No' in black, 'Yes' in red
plot(Heart.train[c('MaxHR', 'Chol')], col=colahd)  ## plot two features
```

**The `cost` parameter** in `svm()` is not the $B$ in (9.15) and Figure 9.7. It has the opposite effect. It is the penalty for total hinge loss. The higher `cost` the higher penalty on total hinge loss. Intuitively, a small `cost` makes the cost of misclassification low, and is often associated with more support vectors.

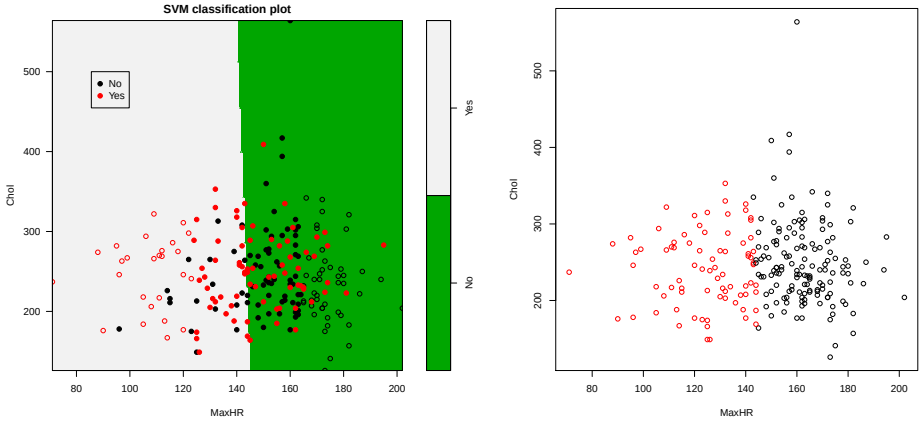
By default, `cost=1`, `scale=TURE` (standardize all input variables), `gamma=1/p` (where $p$ is the number of input variables). For `kernel='linear'`, the `gamma` value is irrelevant.

For the purpose of graphical demonstration, we only use two features to predict the outcome.

```
library(e1071)
svmfit1 = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='linear')
svmfit1
```

In `plot.svm()`, `col=` or `color.palette=` controls the background color for the SVM classifier, `symbolPalette=` controls the colors of plot symbols, and `grid=` controls the smoothness of the classifier boundaries. The left figure is from `plot.svm()`. The right figure shows the fitted values, which are just a dichotomization along the hyperplane.

```
plot(svmfit1, Heart.train, Chol~MaxHR)  ## the default plot (not good-looking)
plot(svmfit1, Heart.train, Chol~MaxHR, grid=200, svSymbol=19, dataSymbol=1,
     color.palette=terrain.colors)  ## left figure
legend(80, 500, col=1:2, pch=19, legend=c('No','Yes'))
##plot(svmfit1, Heart.train, Chol~MaxHR, grid=200, svSymbol=19, dataSymbol=1,
     col=c(1,2), symbolPalette=c(3,4))
plot(Heart.train[c('MaxHR', 'Chol')], col=(svmfit1$fitted=='Yes')+1)  ## right figure
```



We used the default `cost=1`, which probably is not an optimal choice. We should use cross-validation to choose the appropriate value.

```
table(Heart.train$AHD, svmfit1$fitted)                ## training set error rate 62/220 = 0.28
table(Heart.test$AHD, predict(svmfit1, Heart.test))   ## test set error rate 35/83 = 0.42
```

We can extract the raw pieces of information about the support vectors in our SVM model.

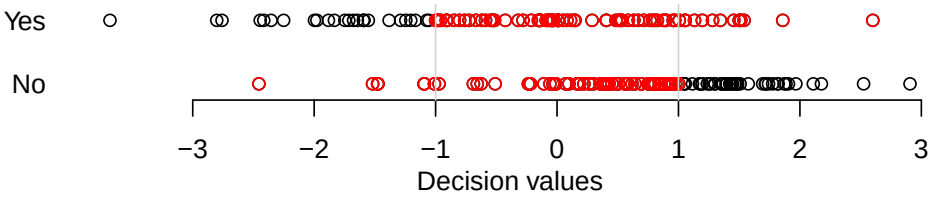
```
names(svmfit1)
svmfit1$index          ## index positions of the support vectors
Heart.train$AHD[svmfit1$index]
svmfit1$SV             ## the support vectors (after scaling) (see next code chunk)

svmfit1$coefs          ## their alpha coefficients in formula (1)
sum(svmfit1$coefs)     ## sum is 0 as expected (not exactly 0 due to rounding errors)
beta = drop(t(svmfit1$coefs) %*% svmfit1$SV)  ## compute beta using formula (1)
beta0 = -svmfit1$rho   ## beta0

sqrt(sum(beta^2))      ## |beta| = 1.15, not 1
```

The decision values are $f_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \hat{\beta}_2 x_{2i}$. A positive decision value results in a prediction of the first category of the outcome, which is the "No" class for the outcome `AHD` in the `Heart` dataset.

```
table(svmfit1$fitted, svmfit1$decision.values > 0)

X = as.matrix(Heart.train[c('MaxHR', 'Chol')])
Xscaled = scale(X)
fi = beta0 + Xscaled %*% beta                                ## fi = beta0 + beta1*x1 + beta2*x2
all.equal(fi, svmfit1$decision.values, check.attributes=F)   ## True
all.equal(svmfit1$SV, Xscaled[svmfit1$index, ])              ## True
```

In the "No" class, the decision values should tend to be positive, and the support vectors are those with $f_i \leq 1$. In the "Yes" class, the decision values should tend to be negative, and the support vectors are those with $f_i \geq -1$. We can check this by plotting the decision values against the class labels.

```
plot(fi, Heart.train$AHD, xlab='', ylab='', yaxt='n', bty='n', ylim=c(.8,2.2))
points(fi[svmfit1$index], Heart.train$AHD[svmfit1$index], col=2)  ## support vectors in red
abline(v=c(-1,1), col='lightgrey')
axis(2, at=1:2, labels=c('No','Yes'), las=1, tick=F)
mtext('Decision values', 1, 2)
```

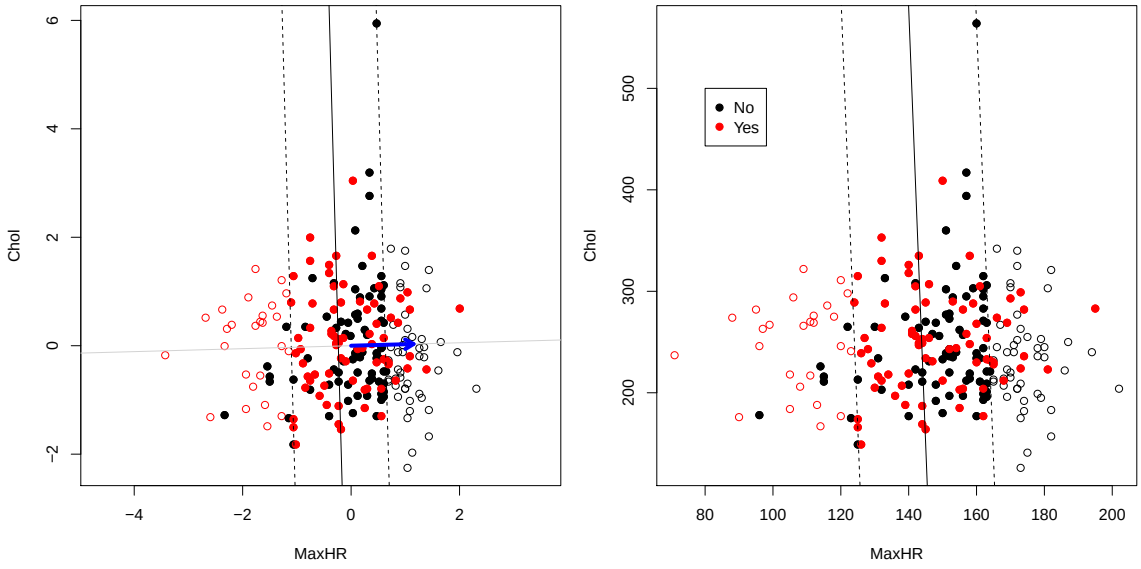



*Hyperplane on the standardized scales of the features*: By default, `scale=T`, and the SVM model is fit to the input variables on their standardized scales. On a 2-dimensional feature space, the hyperplane is $\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = 0$, and the margin boundaries are $\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 = \pm 1$. The hyperplane can be rewritten as $x_2 = -\frac{\hat{\beta}_0}{\hat{\beta}_2} - \frac{\hat{\beta}_1}{\hat{\beta}_2} x_1$. We can plot the hyperplane, margin lines, and the projection direction (the plot below on the left).

```
plot(Xscaled, col=colahd, asp=1)
points(svmfit1$SV, col=colahd[svmfit1$index], pch=19)  ## support vectors in solid dots
abline(-beta0/beta[2], -beta[1]/beta[2])               ## separating hyperplane
abline((-beta0-1)/beta[2], -beta[1]/beta[2], lty=2)    ## margin boundaries
abline((-beta0+1)/beta[2], -beta[1]/beta[2], lty=2)
abline(0, beta[2]/beta[1], col='lightgrey')            ## direction of the projection
arrows(0,0,beta[1],beta[2], col=4, lwd=4, length=.1)   ## show the beta vector
```

*Hyperplane on the original scales of the features*: Let $x_1 = (a_1 - m_1)/s_1$, where $a_1$ is the first input variable on its original scale (with mean $m_1$ and standard deviation $s_1$). Similarly, let $x_2 = (a_2 - m_2)/s_2$. Then the hyperplane expressed on the original scales is $\beta_0 + \beta_1(a_1 - m_1)/s_1 + \beta_2(a_2 - m_2)/s_2 = 0$. It can be rewritten as $a_2 = b_0 + b_1 a_1$, where $b_1 = -\frac{s_2\beta_1}{s_1\beta_2}$ and $b_0 = m_2 - \frac{s_2\beta_0}{\beta_2} - b_1 m_1$. This is plotted below on the right.

```
mm = attr(Xscaled, "scaled:center")   ## the means used in scaling
ss = attr(Xscaled, "scaled:scale")    ## the SDs used in scaling
b1 = -ss[2]*beta[1]/(ss[1]*beta[2])
b0 = mm[2] - ss[2]*beta0/beta[2] - b1*mm[1]

plot(X, col=colahd)
points(X[svmfit1$index, ], col=colahd[svmfit1$index], pch=19)
abline(b0, b1)
abline(mm[2] - ss[2]*(beta0-1)/beta[2] -b1*mm[1], b1, lty=2)
abline(mm[2] - ss[2]*(beta0+1)/beta[2] -b1*mm[1], b1, lty=2)
legend(80, 500, col=1:2, pch=19, legend=c('No','Yes'))
```




**Selection of hyperparameter `cost`**: We can use `tune()` or `tune.svm()` from the `e1071` package to perform cross-validation to select `cost`. The control parameters can be specified through `tune.control()`. The `tune()` function requires the data have no missing data for all variables, even those not used in the SVM model; hence we have to create `Heart0` for the variables of interest. Due to a small sample size, cross-validation does not yield stable results, and the "best" value for `cost` can vary greatly from one run to another.

```
Heart0 = Heart.train[c('AHD', 'MaxHR', 'Chol')]
svmtune1 = tune(svm, AHD ~ MaxHR + Chol, data=Heart0, kernel='linear',
                ranges = list(cost = 2^(-2:12)),
                tunecontrol = tune.control(nrepeat=5, cross=10))

plot(svmtune1, transform.x=log2)  ## not helpful
with(svmtune1$performances, plot(log2(cost), error, type='b'))

names(svmtune1)
svmtune1$performances      ## CV performance
svmtune1$best.parameters
svmtune1$best.model
table(Heart0$AHD, svmtune1$best.model$fitted)                    ## training set; 59/220 = 0.27
table(Heart.test$AHD, predict(svmtune1$best.model, Heart.test))  ## test set; 34/83 = 0.41
```

Note that `train()` in the `caret` package and `svmpath()` in the `svmpath` package can perform a similar task.

**Missing data** in `svm()`: Observations with missing data for the relevant variables are removed by default. The `index` values are the positions of support vectors *after* removal of missing data. This can be confusing.

We now fit an SVM model with all input variables. The `Heart` dataset has 6 observations with missing data.

```
svmfit2 = svm(AHD ~ ., data=Heart.train, kernel='linear')
svmfit2
svmfit2$na.action    ## those that were removed by svm() due to NA
all.equal(rownames(svmfit2$SV), rownames(Heart.train)[svmfit2$index])  ## many mismatches
all.equal(rownames(svmfit2$SV),
          rownames(Heart.train[-svmfit2$na.action,])[svmfit2$index])   ## match now
length(svmfit2$fitted); length(predict(svmfit2, Heart.train))          ## 214 (not 220)
```

This second SVM model clearly performs better than the first one.

```
table(Heart.train$AHD[-svmfit2$na.action], svmfit2$fitted)  ## training set error 26/214 = 0.12
table(Heart.test$AHD, predict(svmfit2, Heart.test))         ## test set error 15/83 = 0.18
```

When there are more than two features, 2-D SVM plots can be misleading. For example,

```
plot(svmfit2, Heart.train, MaxHR ~ Chol)
plot(svmfit2, Heart.train, MaxHR ~ RestBP)
```

**Categorical predictors** in `svm()`: In the `Heart` dataset, there are 13 features, but there are 17 predictors in the SVM model.

```
colnames(svmfit2$SV)  ## 17 columns
svmfit2$scaled        ## whether the columns are scaled
```

They come from a call to `model.matrix()` without intercept. By default, the first categorical variable is coded with dummy variables, one for each category, and the other categorical variables are coded as orthogonal polynomials using `contr.poly()`. While numerical variables are standardized, categorical variables are not rescaled.

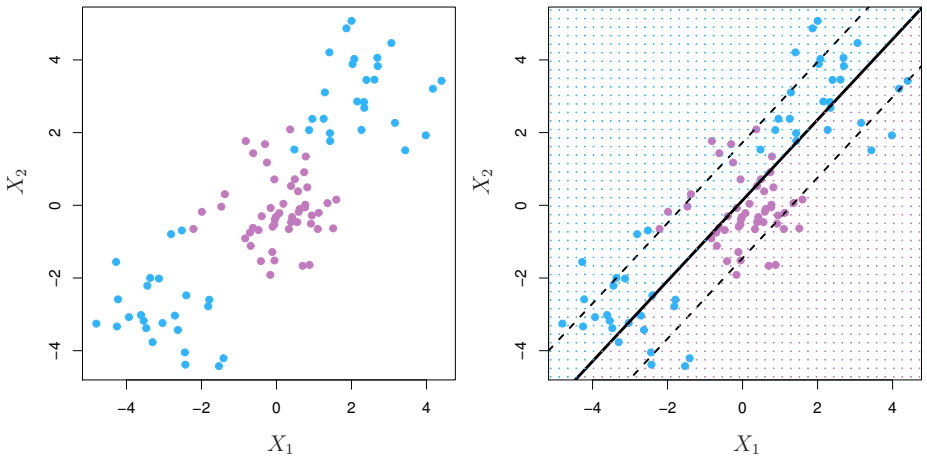
```
tt = model.matrix(AHD ~ . - 1, data=Heart.train)  ## without intercept
dim(tt)       ## 214 x 17 (Observations with missing data are removed!)
colnames(tt)  ## 17 columns; exactly the same names as in colnames(svmfit2$SV)

## same values for support vectors
with(svmfit2, all.equal(SV[, scaled], scale(tt)[index, scaled])) ## True; numerical features
with(svmfit2, all.equal(SV[,!scaled], tt[index, !scaled]))       ## True; categorical features

aa = cbind(rep(1,3), contr.poly(3))  ## contr.poly()
t(aa) %*% aa                         ## the columns in aa are orthonormal

## show how the encoding align with the original values
table(tt[,3],  Heart.train$ChestPain[-svmfit2$na.action])
table(tt[,4],  Heart.train$ChestPain[-svmfit2$na.action])
table(tt[,5],  Heart.train$ChestPain[-svmfit2$na.action])
table(tt[,6],  Heart.train$ChestPain[-svmfit2$na.action])
table(tt[,16], Heart.train$Thal[-svmfit2$na.action])
table(tt[,17], Heart.train$Thal[-svmfit2$na.action])
```

Now we tune this second SVM model using `tune()`:

```
svmtune2 = tune(svm, AHD ~ ., data=Heart.train, kernel='linear',
                ranges = list(cost = 2^(-2:12)))
svmtune2$performances
svmtune2$best.parameters
svmtune2$best.model
with(svmtune2$best.model, table(Heart.train$AHD[-na.action], fitted)) ## training; 24/214 = 0.11
table(Heart.test$AHD, predict(svmtune2$best.model, Heart.test))       ## test; 16/83 = 0.19
```

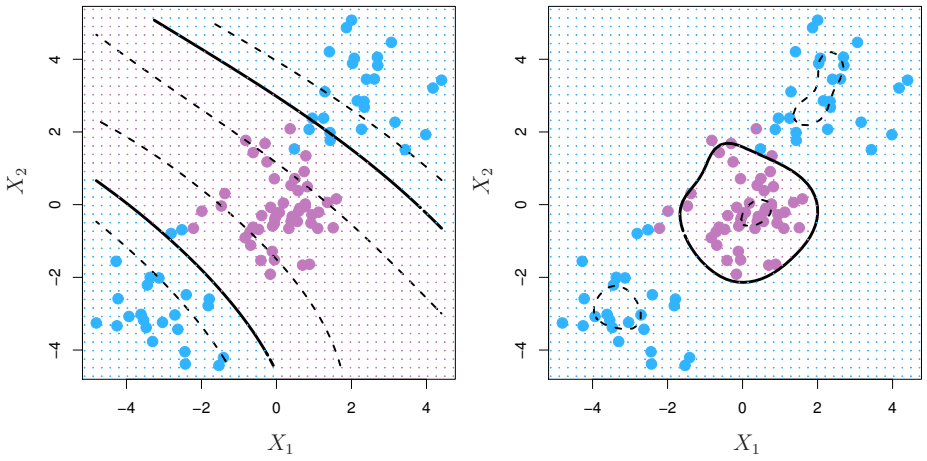
## 11.4 Kernel SVMs (ISLR 9.3)

When the boundary is nonlinear, linear separation will not work (Figure 9.8)




We may

- Introduce more features.
  - Figure 9.9 left: linear separation in the 10-dimensional space $(1, x_1, x_2, x_1^2, x_2^2, x_1x_2, x_1^3, x_2^3, x_1x_2^2, x_1^2x_2) \in R^{10}$ (or the 4-dimensional space $(x_1^3, x_2^3, x_1x_2^2, x_1^2x_2) \in R^4$; unclear from the text)
  - This effectively uses a **polynomial kernel** of degree 3.
- Use kernel functions.
  - A kernel implicitly reflects pairwise "similarities" among observations in a higher dimensional space.
  - Figure 9.9 right: use a **radial kernel**.




A **kernel function** is a symmetric function $K : S \times S \to R$, where $S$ is the feature space. When a kernel function is applied to a dataset of size $n$, the resulting $n \times n$ matrix $\mathbf{K} = \{K(x_i, x_j)\}$ is called the **gram matrix**. The gram matrix is a matrix of *pairwise similarities* of features between observations.

In **kernel SVM**, given a kernel function $K$, we consider the model

$$f(x) = \alpha_0 + \sum_{j=1}^{n} \alpha_j K(x_j, x),$$

with $n + 1$ parameters: $\alpha_0, \alpha_1, \cdots, \alpha_n$. The value of $f(x)$ at $x_i$ is $f_i = f(x_i) = \alpha_0 + \sum_j K(x_i, x_j)\alpha_j$. We use the hinge loss $L(y_i, f_i) = (1 - y_i f_i)_+$ and solve the following optimization problem:

$$\min_{\alpha_0, \alpha} \sum_i (1 - y_i f_i)_+ + \lambda \alpha' \mathbf{K} \alpha, \tag{5}$$

where $\lambda \geq 0$ and $\alpha = (\alpha_1, \ldots, \alpha_n)$. The fitted model is $\hat{f}(x) = \hat{\alpha}_0 + \sum_j K(x, x_j)\hat{\alpha}_j$. If $\hat{\alpha}_j \neq 0$, observation $j$ is a **support vector** because it is involved in defining the model.

- Kernel SVMs share some similarities with smoothing splines. Both start with a model with many ($n$ or $n+1$) parameters, which is then regularized to reduce complexity.

**Linear kernel**: $K(x, x') = \langle x, x' \rangle$, the inner product.

Under the inner product kernel function, it can be shown that the model is

$$f(x) = \alpha_0 + \sum_j \langle x, x_j \rangle \alpha_j = \beta_0 + \langle x, \beta \rangle = \beta_0 + \beta^T x,$$

where $\beta = \sum_j \alpha_j x_j$ and $\beta_0 = \alpha_0$, and that $\|\beta\|_2^2 = \beta'\beta = \alpha' \mathbf{K} \alpha$. Thus, the linear SVM in (3) is the kernel SVM (5) with the inner product kernel.

The `linear` kernel is effectively a polynomial kernel with `degree=1`, `gamma=1`, and `coef0=0`.

```
svmfit1  = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='linear')
svmfit1b = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='polynomial',
               degree=1, gamma=1, coef0=0)
all.equal(svmfit1$fitted, svmfit1b$fitted)                    ## True
all.equal(svmfit1$decision.values, svmfit1b$decision.values)  ## True
```

**Polynomial kernel**: $K(x, x') = (a_0 + \gamma \langle x, x' \rangle)^d$, with hyperparameters $d > 0$ (`degree`), $\gamma$ (`gamma`), $a_0$ (`coef0`).

Polynomial expansion of features: Let $\boldsymbol{x} = (x_1, \ldots, x_p)$ and $\boldsymbol{y} = (y_1, \ldots, y_p)$. Then $K(\boldsymbol{x}, \boldsymbol{y}) = (a_0 + \gamma \langle \boldsymbol{x}, \boldsymbol{y} \rangle)^d = (a_0 + \gamma x_1 y_1 + \ldots + \gamma x_p y_p)^d$. For example, when $p = 2$ and $d = 2$,

$$K(\boldsymbol{x}, \boldsymbol{y}) = a_0^2 + 2a_0\gamma x_1 y_1 + 2a_0\gamma x_2 y_2 + 2\gamma^2 x_1 y_1 x_2 y_2 + \gamma^2 x_1^2 y_1^2 + \gamma^2 x_2^2 y_2^2 = \langle h(\boldsymbol{x}), h(\boldsymbol{y}) \rangle,$$

where $h(\boldsymbol{x}) = (a_0, \sqrt{2a_0\gamma}x_1, \sqrt{2a_0\gamma}x_2, \sqrt{2}\gamma x_1 x_2, \gamma x_1^2, \gamma x_2^2)$ is a 2-degree polynomial expansion from the original feature space $(x_1, x_2) \in R^2$ to a higher dimensional space $R^6$. In general, a kernel SVM with a polynomial kernel is effectively a linear SVM in an **expanded feature space** with all polynomial terms of $\leq d$ degrees of the coordinates. The dimension of the expanded space is $\binom{p+d}{d}$.

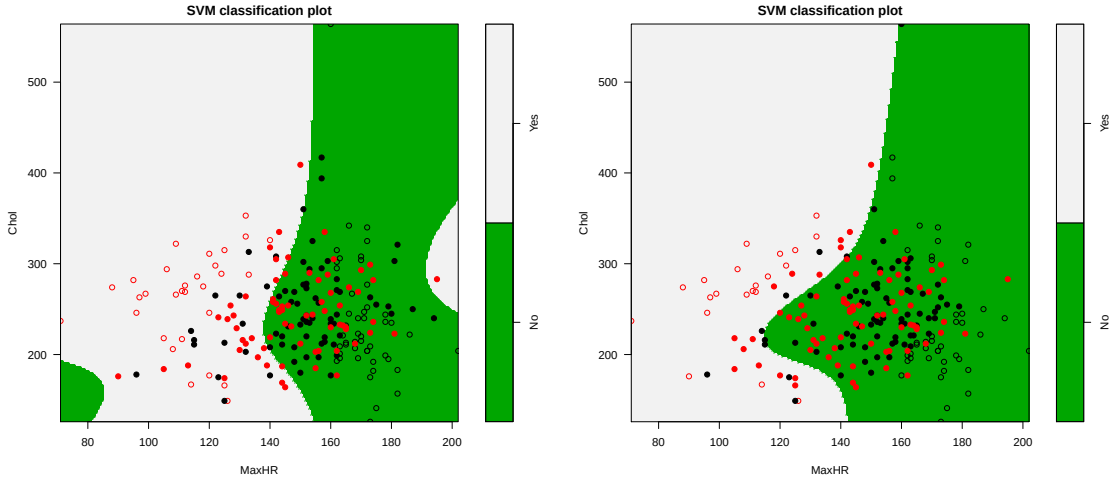
By default, `coef0=0` in `svm()`. The consequence of this default is that the expanded feature space consists of only $d$-degree polynomial terms. To see this, in the example above, when $a_0 = 0$, $K(\boldsymbol{x}, \boldsymbol{y}) = 2\gamma^2 x_1 y_1 x_2 y_2 + \gamma^2 x_1^2 y_1^2 + \gamma^2 x_2^2 y_2^2$, and $h(\boldsymbol{x}) = \gamma(\sqrt{2}x_1 x_2, x_1^2, x_2^2)$ is an expansion with only 2-degree terms. When $p = 2$ and $d = 3$, $a_0 = 0$ corresponds to an expansion from $(x_1, x_2)$ to $(x_1^3, x_2^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2)$, a 4-D space, not the 10-D space $(1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, x_1^3, x_2^3, x_1 x_2^2, x_1^2 x_2)$.

In `svm()` the default values are `cost=1`, `degree=3`, `gamma=1/p`, and `coef0=0`. We first fit the SVM model with the polynomial kernel of degree 3 on `MaxHR` and `Chol`. This is effectively a linear SVM in a 10-D space.

```
svmfit3 = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='polynomial',
              degree=3, gamma=1, coef0=1)
svmfit3
table(Heart.train$AHD, svmfit3$fitted)   ## training set error 59/220
plot(svmfit3, Heart.train, Chol~MaxHR, grid=200, svSymbol=19, dataSymbol=1,
     color.palette=terrain.colors)       ## left figure
```

Using only 3-degree terms (i.e., leaving `coef0=0`). This is effectively a linear SVM in a 4-D space.

```
svmfit3b = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='polynomial', degree=3)
svmfit3b
table(Heart.train$AHD, svmfit3b$fitted)  ## training set error 72/220
plot(svmfit3b, Heart.train, Chol~MaxHR, grid=200, svSymbol=19, dataSymbol=1,
     color.palette=terrain.colors)       ## right figure
```




We now tune it, only on the hyperparameters `cost` and `degree`.

```
Heart0 = Heart.train[c('AHD', 'MaxHR', 'Chol')]
svmtune3 = tune(svm, AHD ~ MaxHR + Chol, data=Heart0, kernel='polynomial', gamma=1, coef0=1,
                ranges=list(cost=4^(-2:6), degree=2:5))
svmtune3$performances
svmtune3$best.parameters
svmtune3$best.model
table(Heart0$AHD, svmtune3$best.model$fitted)  ## confusion matrix
```

Now we fit a polynomial kernel SVM model with all the input variables:

```
Heart.train2 = na.omit(Heart.train)
svmfit4 = svm(AHD ~ ., data=Heart.train2, kernel='polynomial', degree=3, gamma=1, coef0=1)
svmfit4
table(Heart.train2$AHD, svmfit4$fitted)  ## training set error rate 0/214
table(Heart.test$AHD, predict(svmfit4, Heart.test))  ## test set error rate 23/83
```

We tune it, only on the hyperparameters `cost` and `degree`.

```
svmtune4 = tune(svm, AHD ~ ., data=Heart.train2, kernel='polynomial', gamma=1, coef0=1,
                ranges=list(cost=4^(-2:6), degree=2:5))
svmtune4$performances
svmtune4$best.parameters
svmtune4$best.model
table(Heart.train2$AHD, svmtune4$best.model$fitted)  ## training; 6/214
table(Heart.test$AHD, predict(svmtune4$best.model, Heart.test))  ## test set; 25/83
```

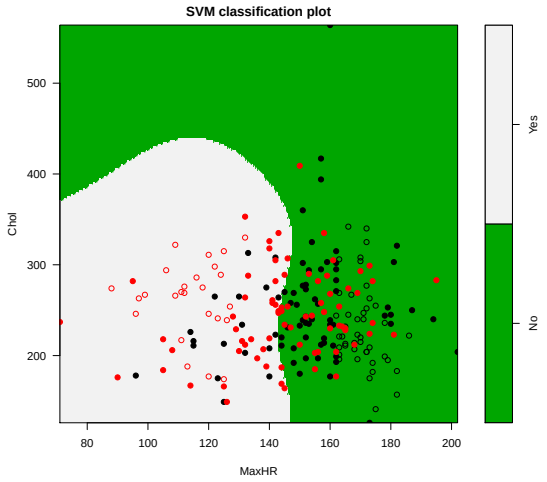
**Radial kernel**: $K(x, x') = e^{-\gamma \|x - x'\|_2^2}$, with hyperparameter $\gamma > 0$ (`gamma`).

It is the default kernel in `svm()`. It is also called the **Gaussian radial kernel** because it effectively reflects a $p$-dimensional Gaussian density when $x' \sim N_p(x, \frac{1}{2\gamma} I_p)$. It is also called the **RBF kernel** (radial basis function kernel). The hyperparameter $\gamma$ controls how local it is when computing pairwise similarity; the higher $\gamma$ the more local similarity. In `svm()`, the default values for the hyperparameters are `cost=1` and `gamma=1/p`.

We first fit an SVM model with the radial kernel on `MaxHR` and `Chol`, leaving `cost` and `gamma` at their default values. Then we tune the model over a range of `cost` and a range of `gamma`.

```
svmfit5 = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='radial')
svmfit5
table(Heart.train$AHD, svmfit5$fitted)  ## training set; 56/220
plot(svmfit5, Heart.train, Chol~MaxHR, grid=200, svSymbol=19, dataSymbol=1,
     color.palette=terrain.colors)
```

```
svmtune5 = tune(svm, AHD ~ MaxHR + Chol, data=Heart0, kernel='radial',
                ranges=list(cost=4^(-2:6), gamma=2^(-3:3)))
svmtune5$best.parameters
svmtune5$best.model
table(Heart0$AHD, svmtune5$best.model$fitted)  ## training set; 56/220
```




We now fit an SVM model with the radial kernel using all the input variables.

```
svmfit6 = svm(AHD ~ ., data=Heart.train2, kernel='radial')
svmfit6
table(Heart.train2$AHD, svmfit6$fitted)  ## training set; 19/214

svmtune6 = tune(svm, AHD ~ ., data=Heart.train2, kernel='radial',
                ranges=list(cost=4^(-2:6), gamma=2^(-3:3)))
svmtune6$best.parameters
svmtune6$best.model
table(Heart.train2$AHD, svmtune6$best.model$fitted)  ## training set; 15/214
```

**Sigmoid kernel**: $K(x, x') = \tanh(a_0 + \gamma\langle x, x'\rangle)$, with hyperparameters $\gamma$ (`gamma`), $a_0$ (`coef0`).

## 11.5 More than 2 classes (ISLR 9.4)

When applying SVM to an outcome with $K$ class ($K > 2$):

- **One-vs-one classification**: Fit a SVM model on each of the $\frac{K(K-1)}{2}$ pairs of classes. Each model only requires data for the two classes being analyzed. The final prediction at any $x_0$ is the majority vote of the predictions at $x_0$ from the $\frac{K(K-1)}{2}$ models. This is how `svm()` does it.
- **One-vs-all classification**: Fit $K$ SVM models, each on one class (coded as 1) versus the other classes (coded as $-1$). At any $x_0$, each SVM model gives a decision value for its class. The class with the highest decision value is the final predicted class at $x_0$.

In the `iris` dataset in base R, the outcome `Species` has three categories. We first fit a linear SVM model. The `probability=T` option allows for probability prediction in addition to the majority vote prediction.

```
library(e1071)
names(iris); dim(iris)  ## 150x5
table(iris$Species)

model1 = svm(Species ~ ., data=iris, kernel='linear', probability=T)
table(iris$Species, model1$fitted)
```

```
names(model1)
dim(model1$decision.values)       ## decision values for pairwise models
colnames(model1$decision.values)  ## names of the feature pairs
model1$nSV                        ## #SVs in each class
```

Our of curiosity, I check if the majority vote from pairwise SVM models would be the fitted category in this model.

```
tmp = model1$decision.values
indivfitted = matrix(,150,3)
for(i in 1:3) indivfitted[,i] = unlist(strsplit(colnames(tmp)[i], '/'))[2 - (tmp[,i] > 0)]
majorityvote = apply(indivfitted, 1, function(x) names(which.max(table(x))))
table(model1$fitted, majorityvote)  ## They agree!
```

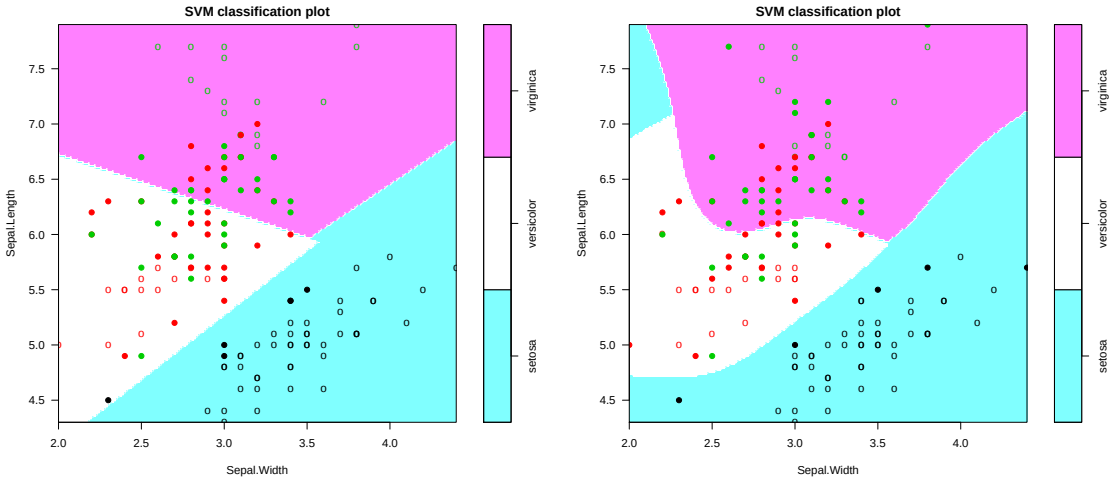
The probability predictions and associated class probabilities can be obtained by calling `predict()` with the `probability=T` option. The class with the largest class probability is the predicted class. The probability prediction may be different from the majority vote prediction. The decision values may reveal why.

```
pred_prob = predict(model1, iris, probability=T)
pred_prob
str(pred_prob)   ## the class probabilities are stored as an attribute
table(pred_prob, apply(attr(pred_prob, 'probabilities'), 1, which.max))  ## agree by defition

pred_prob == model1$fitted           ## different predicted classes for obs 78
attr(pred_prob, 'probabilities')[78,]
model1$decision.values[78,]
model1$fitted[78]
```

Note that the probability modeling in `svm()` has some randomness in its process, and the probability predictions can differ from one run to another.

For SVM models with only two features, we can plot them to see the boundaries.

```
model2 = svm(Species ~ Sepal.Length + Sepal.Width, data=iris, kernel='linear')
plot(model2, iris, Sepal.Length ~ Sepal.Width, grid=200, svSymbol=19)
model2$nSV                        ## #SVs in each class

model3 = svm(Species ~ Sepal.Length + Sepal.Width, data=iris, kernel='radial', cost=10)
plot(model3, iris, Sepal.Length ~ Sepal.Width, grid=200, svSymbol=19)
model3$nSV                        ## #SVs in each class
```



## 11.6 Notes

R packages for SVM:

- `e1071` relies on libsvm and liblinear.
- `kernlab` allows users to define their own kernels.
- `klaR` relies on SVMlight, a C implementation of SVM.
- `svmpath` provides a function to calculate the whole path as a function of `cost`.
- `caret` (document) provides an interface for using `e1071` and `kernlab` packages. Details are here.
- Support Vector Machines in R: a 2006 article reviewing SVM implementation in R.

libsvm and liblinear are C libraries widely used as the backend of many SVM packages, including the Python `sklearn` package.

**Milestones of SVM**:

- Boser, Guyon, Vapnik (1992, Proc COLT) developed the kernel trick to maximum-margin hyperplanes.
- Cortes and Vapnik (1993; 1995, Mach Learning) on soft margin SVM (2008 ACM Paris Kanellakis Award);
- Platt (1998) on sequential minimal optimization (SMO);

**Related events**:

- Vapnik and Chervonenkis (1974) opened the field of "statistical learning theory".
- Vapnik received the 2017 IEEE John von Neumann Medal for his contributions.

## 11.7 Assignment

1. **Homework**: ISLR Chapter 9 Exercises 8
2. Reading for next lecture: NNDL 1