10 PQHS 471 Notes Week 10

10.1 Week 10 Day 1

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 coded as $\{-1, 1\}$. In practice, we often need to code the outcome as a factor. (In `svm()`, a numerical outcome triggers regression SVM unless `type=` is set.)

10.1.1 ISLR 9.1

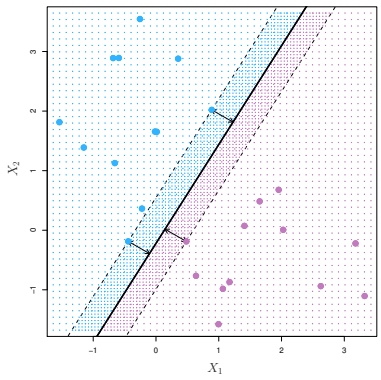
A **hyperplane** in a p-dimensional space is a $(p-1)$-dimensional linear subspace. It can always be represented as

$$\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = \beta_0 + \langle \beta, x \rangle = 0, \tag{9.2}$$

where $\beta = (\beta_1, \ldots, \beta_p)$ and $x = (x_1, \ldots, x_p)$.

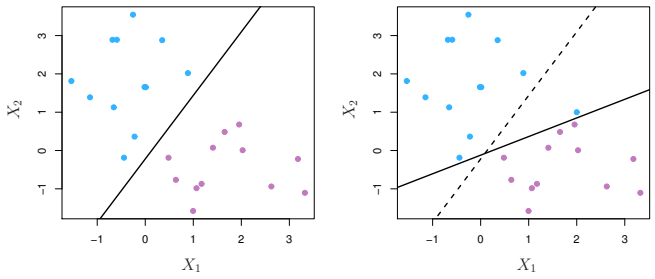
Hyperplane viewed from the perspective of projection: (9.2) can be written as $\langle \beta, x \rangle = -\beta_0$, or equivalently, $\frac{1}{\|\beta\|}\langle \beta, x \rangle = -\frac{\beta_0}{\|\beta\|}$. Here, the left hand side is the projected value of x when it is projected onto the direction of β. Thus, the hyperplane as defined in (9.2) consists of those x that have the same projected value $-\frac{\beta_0}{\|\beta\|}$, and **the hyperplane is perpendicular to** β.

Now consider binary data that can be *perfectly separated* in the feature space. For every separating hyperplane, $\beta_0 + \langle \beta, x \rangle = 0$ with $\|\beta\| = 1$, $\beta_0 + \langle \beta, x_i \rangle > 0$ for all observations in one class (say, those with $y_i = 1$) and $\beta_0 + \langle \beta, x_i \rangle < 0$ for all observations in the other class (say, those with $y_i = -1$). The **margin** is the smallest distance to the separating hyperplane. The *optimal separating hyperplane* is defined as the one with the largest margin, and the corresponding classifier is called the *maximal margin classifier* (Figure 9.3). The observations that define the margin are called **support vectors**. 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, and can be too sensitive (Figure 9.5).
- This will not work when data cannot be perfectly separated.



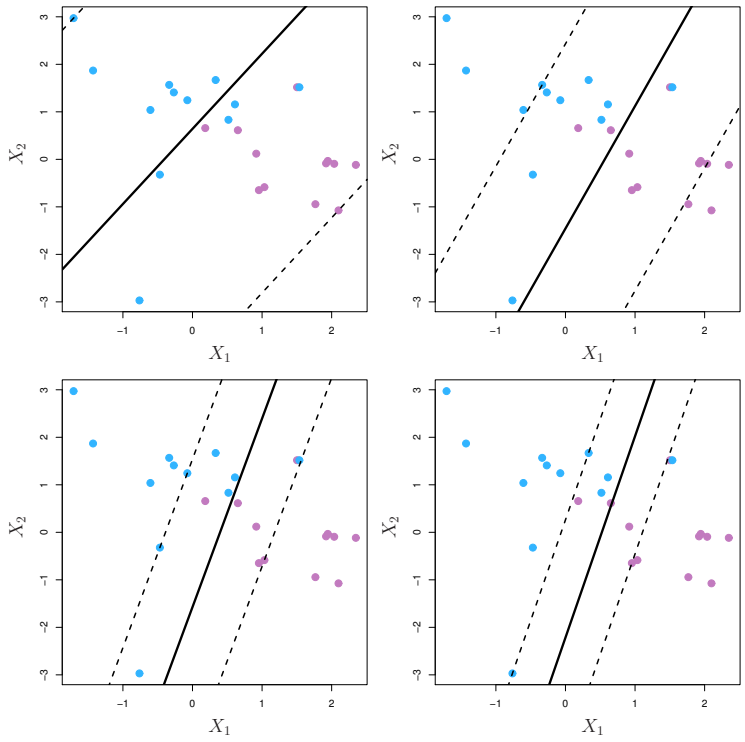
Mathematical definition: Let $f_i = \beta_0 + \langle \beta, x_i \rangle$. The smallest distance to the hyperplane among those with $y_i = 1$ is $\min_{y_i=1} y_i f_i$, and that among those with $y_i = -1$ is $\min_{y_i=-1} y_i f_i$. So the margin is $\min y_i f_i$. The problem can be expressed as

$$\max_{\beta_0, \beta} M \quad \text{subject to } \|\beta\|_2 = 1 \text{ and } y_i(\beta_0 + \langle \beta, x_i \rangle) \geq M. \tag{9.9–9.11}$$

Alternative expression: $\min \|\beta\|_2$ subject to $y_i(\beta_0 + \langle \beta, x_i \rangle) \geq 1$ for all observations.

10.1.2 ISLR 9.2 Support vector classifiers (linear SVMs)

Allow some observations to be on the "wrong" side of the margin. Give "allowances" to observations but control the total "budget", B. Illustrated 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.



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. This is the default in `svm()` (`scale=T`).
- B is a tuning parameter and should be selected using cross-validation. In `svm()`, `cost` is used instead; `cost` has an opposite effect of B (details below).
- A large B tends to lead to a wider margin and more support vectors.
- The observations that define the hyperplane and margin are called **support vectors**. It can be shown that the SVM solution 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.
- Linear SVM is similar to ridged logistic regression (details below).

Mathematical definition:

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

- Here, the ϵ_i's are called **slack variables**.
- All 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.

Let $f_i = \beta_0 + \langle \beta, x_i \rangle$. An alternative expression is:

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

Another alternative expression is:

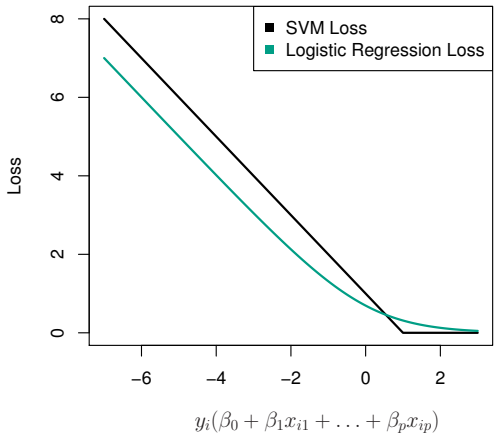
$$\min_{\beta_0, \beta} \sum_i (1 - y_i f_i)_+ + \lambda \|\beta\|_2^2, \tag{2}$$

where $\lambda \geq 0$. This is similar to ridge regression but with $\sum_i (y_i - \hat{y}_i)^2$ replaced by $\sum_i (1 - y_i f_i)_+$. The function $L(y_i, f_i) = (1 - y_i f_i)_+$ is called the **hinge loss**. All support vectors incur some hinge loss. Non-support vectors have zero hinge loss. If other words, 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 $\log(1+e^{-y_i f_i})$ (which is the negative log-likelihood when $f_i = \ln \frac{p}{1-p}$), we have the **ridged logistic regression**,

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

The two loss functions are similar (Figure 9.12). It shows the similarity between linear SVM and ridged logistic regression.



When using `svm()` in the R `e1071` package for classification, make sure the outcome is coded as a factor (otherwise a regression SVM could 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. 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.

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.

```
library(e1071)

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)

set.seed(2018)
```

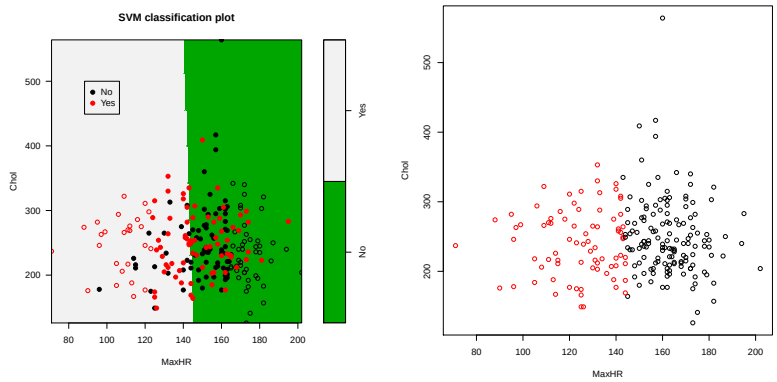
```
trainidx = sample(1:nrow(Heart), 220)
Heart.train = Heart[trainidx, ] ## training set
Heart.test = Heart[-trainidx, ] ## test set

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

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

To plot the model result, `plot.svm()` can be used. `col` or `color.palette` controls the background color for the SVM classifier, while `symbolPalette` controls the colors of plot symbols. I do not know how to add the margin lines to it.

```
plot(svmfit1, Heart.train, Chol~MaxHR)  ## the default gives an ugly plot
plot(svmfit1, Heart.train, Chol~MaxHR, grid=200, svSymbol=19, dataSymbol=1, color.palette=terrain.colors)
legend(80, 500, col=1:2, pch=19, legend=c('No','Yes'))
##plot(svmfit1, Heart.train, Chol~MaxHR, svSymbol=19, dataSymbol=1, col=c(1,2), symbolPalette=c(3,4))
```

The fitted values are just a dichotomization along the hyperplane, as shown in the plot below.

```
svmfit1$fitted
table(svmfit1$fitted, svmfit1$decision.values>0)  ## Predict 'No' if decision.value > 0
table(Heart.train$AHD, svmfit1$fitted)  ## confusion matrix
plot(Heart.train[c('MaxHR', 'Chol')], col=(svmfit1$fitted=='Yes')+1)
```



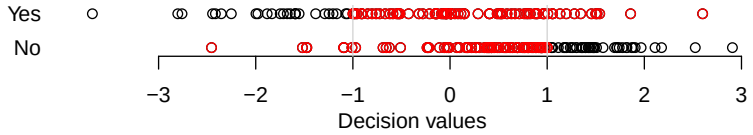
We can extract various information about the SVM model. For example, the raw pieces of information for the support vectors are below:

```
names(svmfit1)
svmfit1$index  ## positions of the support vectors
Heart.train$AHD[svmfit1$index]
svmfit1$coefs  ## their alpha coefficients (the alpha's in (1))
beta = drop(t(svmfit1$coefs) %*% svmfit1$SV)  ## formula (1)
beta0 = -svmfit1$rho

sqrt(sum(beta^2))  ## |beta| = 1 as expected (not exactly 1 due to rounding errors)
sum(svmfit1$coefs)  ## 0 as expected (not exactly 0 due to rounding errors)
```

The decision values are $f_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \hat{\beta}_2 x_{2i}$. 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 $f_i \geq -1$. Below we check if this is correct.

```
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

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)
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)
```

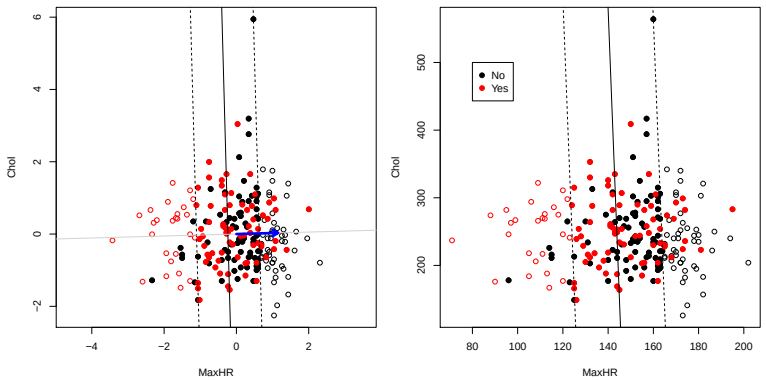


Hyperplane on the standardized scales of the input variables: By default, `scale=T`, and the SVM model is fit to the input variables on their standardized scales. On a 2-dimensional feature space, the hyperplane is $\hat{\beta}_0+\hat{\beta}_1 x_1+\hat{\beta}_2 x_2=0$, and the margin lines are $\hat{\beta}_0+\hat{\beta}_1 x_1+\hat{\beta}_2 x_2=\pm 1$. 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)  ## aps=1 is to display the same unit length for x and y
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 lines
abline((-beta0+1)/beta[2], -beta[1]/beta[2], lty=2)
abline(0, beta[2]/beta[1], col='lightgrey')  ## direction of 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 input variables: Since `scale=T` was used to fit the model, $x_1=(a_1-m_1)/s_1$, where a_1 is the first input variable on its original scale (with mean m_1 and standard deviation s_1). Similarly, $x_2=(a_2-m_2)/s_2$. 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 $a_2=m_2-\frac{s_2\beta_0}{\beta_2}-\frac{s_2\beta_1}{s_1\beta_2}(a_1-m_1)=b_0+b_1 a_1$, 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")
ss = attr(Xscaled, "scaled:scale")
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 cost: We can use `tune()` to perform cross-validation to select `cost`. The function requires the data have no missing data for all variables in the data frame. We create `Heart0` for the variables of interest.

```
Heart0 = Heart.train[c('AHD', 'MaxHR', 'Chol')]
svmtune1 = tune(svm, AHD ~ MaxHR + Chol, data=Heart0, kernel='linear',
                ranges=list(cost=2^(-5:10)))
names(svmtune1)
svmtune1$performances
svmtune1$best.model
table(Heart0$AHD, svmtune1$best.model$fitted)  ## confusion matrix
```

The `train()` function in the `caret` package can perform a similar task. The `svmpath()` function in the `svmpath` package can also do something similar.

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 (which 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$na.action  ## those removed due to NA
all.equal(rownames(Heart.train[-svmfit2$na.action,])[svmfit2$index], rownames(svmfit2$SV))  ## True

table(Heart.train$AHD[-svmfit2$na.action], svmfit2$fitted)  ## training data confusion matrix
table(Heart.test$AHD, predict(svmfit2, Heart.test))  ## test data confusion matrix

all.equal(svmfit2$fitted, predict(svmfit2, Heart.train))  ## True
length(svmfit2$fitted); length(predict(svmfit2, Heart.train))  ## 214 (not 220)
```

When there are more than two input variables, 2-D plots can be misleading.

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

Categorical predictors in `svm()`: For a categorical predictor, dummy variables are used. In the `Heart` dataset, there are 13 input variables, but there are 17 predictors in the model. They come from a call to `model.matrix()` without intercept. In this situation, 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()`. Numerical variables stay the same and are standardized by default. Categorical variables are not scaled.

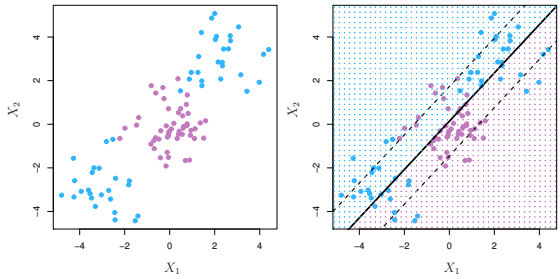
```
colnames(svmfit2$SV)  ## 17 columns
svmfit2$scaled  ## categorical variables are not "scaled"

tt = model.matrix(AHD ~ .-1, data=Heart.train)  ## without intercept
dim(tt)  ## 214 x 17
colnames(tt)  ## 17 columns

all.equal(scale(tt)[svmfit2$index, c(1,2,7:15)], svmfit2$SV[, c(1,2,7:15)])  ## True for numerical variabl
all.equal(tt[svmfit2$index, c(3:6,16:17)], svmfit2$SV[, c(3:6,16:17)])  ## True

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])
contr.poly(3)
```

Now we tune the SVM model using `tune()`:

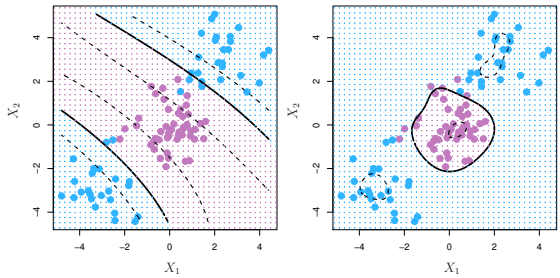
```
svmtune2 = tune(svm, AHD ~ ., data=Heart.train, kernel='linear',
                ranges=list(cost=2^(-5:10)))
svmtune2$performances
svmtune2$best.model
table(Heart.train$AHD[apply(is.na(Heart.train), 1, sum)==0], svmtune2$best.model$fitted)
```

10.1.3 ISLR 9.3 Kernel SVMs

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



- Introduce more features.
  - Figure 9.9 left: linear separation in the 10-D space $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2, x_1^3, x_2^3, x_1x_2^2, x_1^2x_2) \in R^{10}$.
  - Effectively using a *polynomial kernel* of degree 3.
- Use kernel functions, which implicitly represent "distances" in a higher dimensional space.
  - Figure 9.9 right: use a *radial kernel*.



A **kernel function** is 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* among of the n observations. In **kernel SVM**, we solve the following problem:

$$\min_{\alpha_0,\alpha} \sum_i (1 - y_i(\alpha_0 + \mathbf{K}\alpha))_+ + \lambda\alpha'\mathbf{K}\alpha, \tag{3}$$

where $\lambda \geq 0$ and $\alpha = (\alpha_1, \ldots, \alpha_n)$. Here, the prediction model is a function $f(x) = \alpha_0 + \sum_j K(x, x_j)\alpha_j$ with $n + 1$ parameters α_0 and α. The observations with $\hat{\alpha}_j \neq 0$ are the **support vectors**.

When the hinge loss in (3) is replaced by other loss functions, the problem is called the **generalized ridge problem**. (This is somewhat similar to smoothing splines, where we have n parameters and regularize them.)

Linear kernel: When the kernel function is the inner product, $K(x, x') = \langle x, x' \rangle$, $f(x) = \alpha_0 + \sum_j \langle x, x_j \rangle \alpha_j$. It can be shown that $f(x) = \beta_0 + \langle x, \beta \rangle$, 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 kernel SVM (3) with the inner product kernel is the linear SVM in (2), thus the name "linear" kernel.

Polynomial kernel of degree d: $K(x, x') = (a_0 + \gamma\langle x, x' \rangle)^d$

Hyperparameters: In additional to `cost`, there are three more hyperparameters: `degree` (d, the main one), `gamma` (γ), and `coef0` (a_0). 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

## Now vary the values of gamma and coef0 and see how happens
svmfit1c = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='polynomial',
               degree=1, gamma=.01, coef0=-10)
table(svmfit1$fitted, svmfit1c$fitted)
plot(svmfit1$decision.values, svmfit1c$decision.values); abline(h=0,v=0,col='grey')
```

Polynomial expansion of features uses a special case of polynomial kernel, $K(x, x') = (1 + \langle x, x' \rangle)^d$, with $\gamma = 1$ and $a_0 = 1$. This is often presented as the polynomial kernel in textbooks (e.g., ISLR (9.22)). Using coordinates $x = (a_1, \ldots, a_p)$ and $x' = (b_1, \ldots, b_p)$, it is $K(x, x') = (1 + \langle x, x' \rangle)^d = (1 + a_1 b_1 + \ldots + a_p b_p)^d$. For example, when $d = 2$, $K(x, x') = 1 + a_1^2 b_1^2 + \cdots + a_p^2 b_p^2 + 2a_1 b_1 + \cdots + 2a_p b_p + 2\sum_{i<j} a_i b_i a_j b_j = \langle h(x), h(x') \rangle$, where $h : R^p \to R^q$ with $h(x) = (1, a_1^2, \ldots, a_p^2, \sqrt{2}a_1, \ldots, \sqrt{2}a_p, \sqrt{2}a_1 a_2, \sqrt{2}a_1 a_3, \ldots, \sqrt{2}a_{p-1} a_p)$ and $q = \frac{(p+2)(p+1)}{2}$. Thus, fitting a kernel SVM with the kernel $K(x, x') = (1 + \langle x, x' \rangle)^2$ is effectively **expanding the features** to include all 1-degree and 2-degree terms of the features, and then fitting a linear SVM in the expanded feature space.

- In general, fitting a kernel SVM with the kernel $K(x, x') = (1 + \langle x, x' \rangle)^d$ is effectively doing a linear SVM in an expanded feature space with all terms of $\leq d$ degrees. (The dimension is $\binom{p+d}{d}$ with p features.)
- To fit such a polynomial SVM with `svm()`, one needs to specify `gamma=1` and `coef0=1`.
- If we relied on the `svm()` default values of `degree=3`, `gamma=1/p`, and `coef0=0`, we would be using the kernel $K(x, x') = p^{-3}\langle x, x' \rangle^3$, which is effectively feature expansion with only 3-degree terms. For example, when $p = 2$, the expansion would be $h(a_1, a_2) = p^{-3}(a_1^3, a_2^3, \sqrt{3}a_1^2 a_2, \sqrt{3}a_1 a_2^2)$, a 4-D space (instead of a 10-D space as in the example above).

We now fit the SVM model with the polynomial kernel of degree 3 on `MaxHR` and `Chol`:

```
## polynomial expansion of features with degree 3
svmfit3 = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='polynomial',
              degree=3, gamma=1, coef0=1)
svmfit3
table(Heart.train$AHD, svmfit3$fitted)  ## confusion matrix
plot(svmfit3, Heart.train, Chol~MaxHR, grid=200, svSymbol=19, dataSymbol=1, color.palette=terrain.colors)

## using only 3-degree terms
svmfit3b = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='polynomial', degree=3)
table(Heart.train$AHD, svmfit3b$fitted)  ## confusion matrix
plot(svmfit3b, Heart.train, Chol~MaxHR, grid=200, svSymbol=19, dataSymbol=1, color.palette=terrain.colors)

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

We now fit the SVM model with all the input variables:

```
Heart.train2 = Heart.train[apply(is.na(Heart.train), 1, sum)==0,]
svmfit4 = svm(AHD ~ ., data=Heart.train2, kernel='polynomial', degree=3, gamma=1, coef0=1)
svmfit4
table(Heart.train2$AHD, svmfit4$fitted)  ## confusion matrix

## tune it
```

```
svmtune4 = tune(svm, AHD ~ ., data=Heart.train2, kernel='polynomial', degree=3, gamma=1, coef0=1,
                ranges=list(cost=2^(-5:5)))
svmtune4$performances
svmtune4$best.model
table(Heart.train2$AHD, svmtune4$best.model$fitted)  ## confusion matrix
```

Radial kernel: $K(x, x') = e^{-\gamma\|x-x'\|_2^2}$

It is also called the *Gaussian radial kernel* (because it is effectively a one-dimensional Gaussian density for $\|x-x'\|_2$), or the *radial basis function (RBF) kernel*. In additional to `cost`, there is another hyperparameter: `gamma` (γ). γ controls how local we go when calculating pairwise similarity. The higher γ the more local.

We now fit the SVM model with the radial kernel on `MaxHR` and `Chol`, leaving the gamma at its default:

```
svmfit5 = svm(AHD ~ MaxHR + Chol, data=Heart.train, kernel='radial')
svmfit5
table(Heart.train$AHD, svmfit5$fitted)  ## confusion matrix

## tune it
svmtune5 = tune(svm, AHD ~ MaxHR + Chol, data=Heart0, kernel='radial',
                ranges=list(cost=2^(-5:5), gamma=2^(-5:0)))
svmtune5$best.model
table(Heart0$AHD, svmtune5$best.model$fitted)  ## confusion matrix
```

We now fit the 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)  ## confusion matrix

## tune it
svmtune6 = tune(svm, AHD ~ ., data=Heart.train2, kernel='radial',
                ranges=list(cost=2^(-5:5), gamma=2^(-3:0)))
svmtune6$best.model
table(Heart.train2$AHD, svmtune6$best.model$fitted)  ## confusion matrix
```

10.1.4 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 is the majority vote from the $\frac{K(K-1)}{2}$ predictions. This is how `svm()` does it.
- **One-vs-all classification**: Fit K SVM models, each on one class (coded as 1) versus the rest (coded as -1). For prediction on a data point x_0, each model gives a decision value. The class with the highest decision value is the final predicted class.

In the `iris` dataset in base R, the outcome has three categories. We first fit a linear SVM model. The `probability=T` option allows us to generate predicted probabilities when calling `predict()`.

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

model1 = svm(Species ~ ., data=iris, kernel='linear', probability=T)
##model2 = svm(x=as.matrix(iris[,1:4]), y=iris$Species, kernel='linear', probability=T)
table(iris$Species, model1$fitted)
pred_prob = predict(model1, iris, probability=T)
```

```
table(apply(attr(pred_prob, 'probabilities'), 1, which.max), model1$fitted)  ## agreed

names(model1)
dim(model1$decision.values)  ## 3 sets of decision values
colnames(model1$decision.values)  ## 3 sets of decision values
model1$nSV  ## #SVs in each class
```

I do not know how the "decision values" are computed.

```
table(apply(model1$decision.values, 1, which.max), model1$fitted)  ## maximum is not helpful
```

Try this on two input variables:

```
model = svm(Species ~ Sepal.Length + Sepal.Width, data=iris, kernel='linear', probability=T)
table(iris$Species, model$fitted)
pred_prob = predict(model, iris, probability=T)
table(apply(attr(pred_prob, 'probabilities'), 1, which.max), model$fitted)  ## agreed

plot(model, iris, Sepal.Length ~ Sepal.Width, grid=200, svSymbol=19)
```

Notes on R packages for SVM:

- `e1071` uses libsvm and liblinear: C libraries widely used as the backend of many SVM packages (including `sklearn`).
- `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`.
- Support Vector Machines in R: a 2006 article reviewing SVM implementation in R.
- `caret` (document) provides an interface for using `e1071` and `kernlab` packages. Details are here.

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.

10.1.5 Assignment

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

10.2 Week 10 Day 2 (Off; I am out of town)