HDS 5. Multiple regression with growing p

Matti Pirinen, University of Helsinki

12-14.11.2019

So far we have considered statistical inference methods that are based on univariate models. That is, models where we estimate/test only one variable at a time. Such methods are computationally simple and can therefore be applied for very high dimensional problems. However, they ignore dependencies between predictors. To illustrate why this is a problem, consider two predictors x and z that are correlated with correlation $r = 0.7$ and of which only x truly affects outcome y. Let's see what happens when we fit univariate models vs. joint regression model.

To create such variables, let's write $x = u + \varepsilon_x$ and $z = u + \varepsilon_z$, where $\text{Var}(u) = |r|$ and $\text{Var}(\varepsilon_x) = \text{Var}(\varepsilon_z) = 1 - |r|$ with ε_x and ε_z independent of each other and of u. Then

$$\text{Cor}(x,z) = \frac{\text{Cov}(x,z)}{\sqrt{\text{Var}(x)\cdot\text{Var}(z)}} = \frac{\text{Cov}(u+\varepsilon_x, u+\varepsilon_z)}{\sqrt{\text{Var}(u+\varepsilon_x)\cdot\text{Var}(u+\varepsilon_z)}} = \frac{\text{Cov}(u,u)}{\sqrt{(\text{Var}(u)+\text{Var}(\varepsilon_x))(\text{Var}(u)+\text{Var}(\varepsilon_z))}} = \frac{|r|}{\left(\sqrt{|r|}+\sqrt{1-|r|}\right)^2}$$

(If we wanted negatively correlated variables, we would write $z = -u + \varepsilon_z$ instead.)

```
n = 1000
r = 0.7
u = rnorm(n, 0, sqrt(abs(r))) #temporary variable that is used for generating x and z
x = scale(u + rnorm(n, 0, sqrt(1-abs(r))))
z = scale(sign(r)*u + rnorm(n, 0, sqrt(1-abs(r))))
cor(x, z) #check that now cor(x,z) = r
```

```
##           [,1]
## [1,] 0.7036026
```

Let's generate $y = \sqrt{0.2}\cdot x + \mathcal{N}(0, 0.8)$ whence x will explain 20% of variance of y, and $\text{Var}(y) = \sqrt{0.2}^2 + 0.8 = 1$.

```
y = scale(sqrt(0.2)*x + rnorm(n, 0, sqrt(0.8))) # x explains 20% of y
summary(lm(y ~ 0 + x))
```

```
##
## Call:
## lm(formula = y ~ 0 + x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0237 -0.6078 -0.0290  0.6301  2.8429
##
## Coefficients:
##   Estimate Std. Error t value Pr(>|t|)
## x  0.44291    0.02837   15.61   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8966 on 999 degrees of freedom
## Multiple R-squared:  0.1962, Adjusted R-squared:  0.1954
## F-statistic: 243.8 on 1 and 999 DF,  p-value: < 2.2e-16
```

Note that we did not use z to generate y, but z and y are still highly correlated:

$$\text{Cor}(y,z) = \frac{\text{Cov}(y,z)}{\sqrt{\text{Var}(y)\cdot\text{Var}(z)}} = \frac{\text{Cov}(\sqrt{0.2}\cdot u + \sqrt{0.2}\cdot\varepsilon_x + \varepsilon_y, u + \varepsilon_z)}{\sqrt{1\cdot 1}} = \sqrt{0.2}\cdot\text{Cov}(u,u) = \sqrt{0.2}\cdot|r|$$

Thus, we expect the regression coefficient of y on z to be $\sqrt{0.2}\cdot 0.7 \approx 0.31$.

```
summary(lm(y ~ 0 + z)) # z does not have a direct effect on y but will be significant here
```

```
##
## Call:
## lm(formula = y ~ 0 + z)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.06774 -0.66576  0.02115  0.63622  2.81372
##
## Coefficients:
##   Estimate Std. Error t value Pr(>|t|)
## z  0.30243    0.03016   10.03   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9532 on 999 degrees of freedom
## Multiple R-squared:  0.09147,    Adjusted R-squared:  0.09056
## F-statistic: 100.6 on 1 and 999 DF,  p-value: < 2.2e-16
```

Luckily, the joint model of `y ~ x + z` can tease the effects of x and z apart.

```
summary(lm(y ~ 0 + x + z))
```

```
##
## Call:
## lm(formula = y ~ 0 + x + z)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.01014 -0.60618 -0.02546  0.62552  2.84596
##
## Coefficients:
##   Estimate Std. Error t value Pr(>|t|)
## x  0.45573    0.03993  11.412   <2e-16 ***
## z -0.01822    0.03993  -0.456    0.648
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8969 on 998 degrees of freedom
## Multiple R-squared:  0.1963, Adjusted R-squared:  0.1947
## F-statistic: 121.9 on 2 and 998 DF,  p-value: < 2.2e-16
```

We have seen that if we test z univariately, we observe a strong effect on y, but that effect disappears when we do multiple regression of y on both x and z. Thus, regression is able to split the effect among correlated variables appropriately. This is very important in order to separate possible *direct effects* of x on y from *confounders* z, that are simply correlated with both x and y, but not anymore important in explaining y once we have measured predictors with direct effects on y.

Example 5.1. In medical literature the role of HDL and LDL cholesterol levels in the risk of heart disease has been studied a lot over decades. Both are marginally correlated with the risk, HDL negatively and LDL

positively. Recent studies and medical trials suggest that LDL is "causal", in the sense that medication lowering directly LDL levels will reduce disease risk, whereas HDL is not causal in that sense. This is extremely important information for developing preventive treatment for heart disease.

Note that with statistical models alone, we cannot ever *prove* a causal relationship between x and y but we can show that z is unlikely to have a causal relationship if its effect goes away once a better predictor x is found.

If variables are highly correlated, there will be little information how the effect should be split among them and therefore there SEs will be large and effect size estimates will be unstable. So in practice we do not want to include very highly correlated variables (say $|r| > 0.95$, but this threshold is arbitrary and will also depend on the amount of data available) into regression models. We will come back to this point next week when we discuss ridge regression as a way to stabilize models with highly correlated variables.

What about if the predictors were independent of each other? Do we then need to use a joint model to get the effect estimates correct? If x and z are independent, then their marginal effects (from separate models $y = x\beta_x + \varepsilon$ and $y = z\beta_z + \varepsilon$) are estimating the same quantities as the joint model $y = x\gamma_x + z\gamma_z + \varepsilon$, i.e., $\beta_x = \gamma_x$ and $\beta_z = \gamma_z$. However, the joint model is more precise than the marginal models (smaller SEs) in case that both x and z have an effect on y, because the joint model has lower residual variance and hence smaller SEs.

Let's next consider how we can use multiple regression models in higer dimensional (HD) problems. We begin from a problem that results from a naive approach. This example also takes us through some central issues in HD data analysis where we need to balance between the variance and bias of our statistical method in order to avoid **overfitting** but still get useful information out from the model.

READ: *2.2.1 Measuring the Quality of Fit* from book ISL and see slides HDS5.

- Mean squared error (MSE) for measuring fit
- Training and test data sets, training and test MSE
- How training and test MSE behave as a function of model flexibility?
- Overfitting

Overfitting

To make overfitting problem concrete in a linear regression setting, let's consider $n_r = 150$ training individuals that were measured on $p = 70$ variables and an outcome variable y. Four variables, x_1, x_2, x_3, x_4 will be truly associated with y; 66 others are just random noise. Let's build a prediction model for y using linear regression and let's test that model in additional $n_t = 150$ test individuals.

```
set.seed(20102017)
p = 70
n.tr = 150 # training set
n.te = 150 # test set
n = n.tr + n.te
i.tr = 1:n.tr; i.te = (n.tr + 1):n #indexes for training and testing
phi = 0.05 # variance explained by each relevant predictor, should be 0 < phi < 1.
b = rep(c(sqrt(phi / (1-phi)), 0), c(4, p-4)) #effects 1,2,3,4 are non-zero, see Lect. 0.1 for "phi"
X = matrix(rnorm(n*p), nrow = n) #columns 1,...,4 of X have effects, other 66 cols are noise
eps = scale(rnorm(n, 0, 1)) #epsilon, error term
y = scale(X%*%b + eps) #makes y have mean = 1, var = 1
y.tr = y[i.tr]; y.te = y[i.te]
X.tr = data.frame(scale(X[i.tr,])); X.te = data.frame(scale(X[i.te,]))
lm.fit =  lm(y.tr ~ 0 + . , data = X.tr) #fit the model to training data
#(remove intercept because formulas below become easier,
```

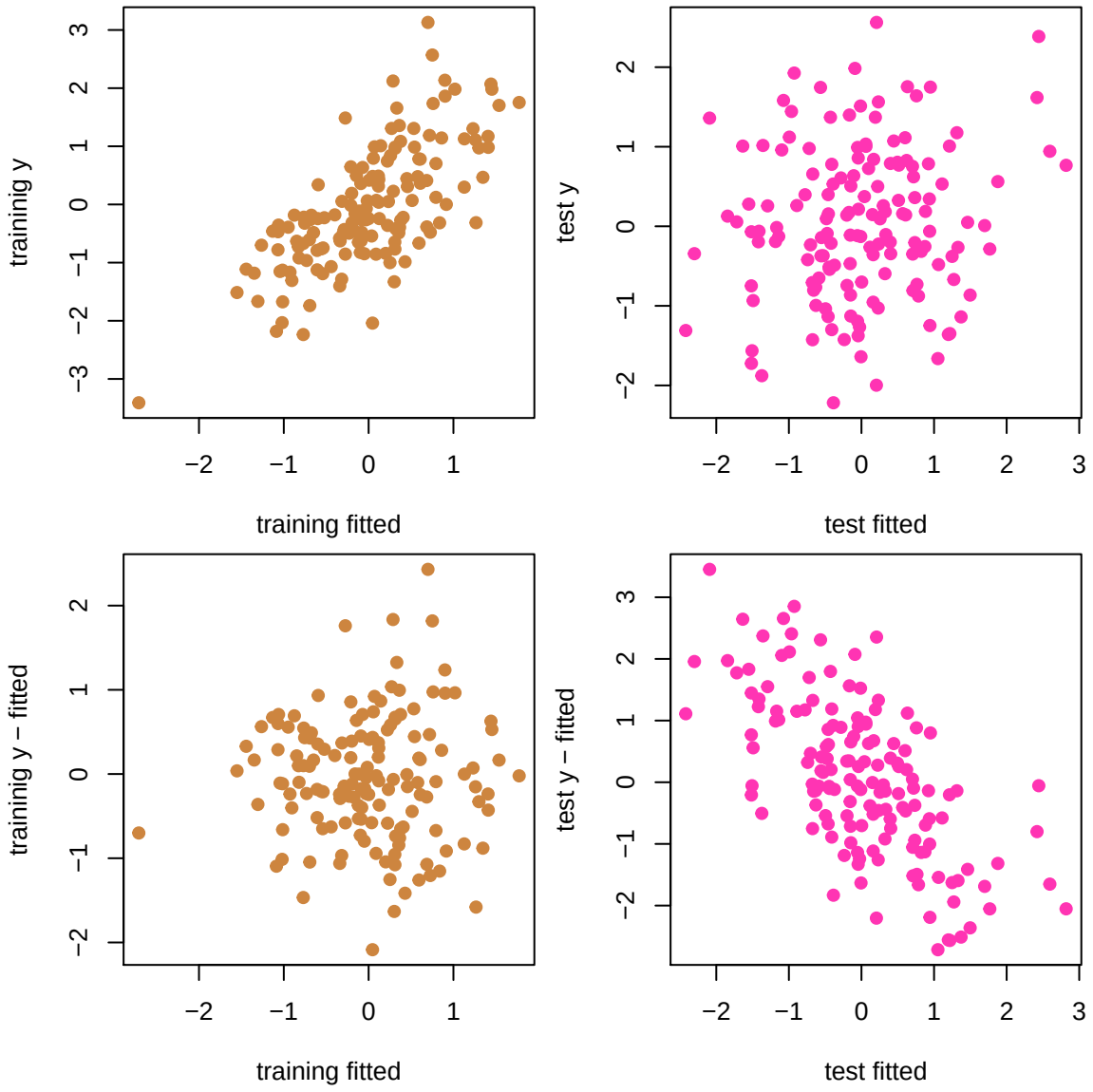
```r
# only possible because cols of X and y have mean = 0.)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = y.tr ~ 0 + ., data = X.tr)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.08623 -0.53466 -0.09445  0.43393  2.43027
##
## Coefficients:
##      Estimate Std. Error t value Pr(>|t|)
## X1   0.229311   0.107648   2.130  0.03623 *
## X2   0.194258   0.123795   1.569  0.12055
## X3   0.260994   0.111172   2.348  0.02136 *
## X4   0.308204   0.113932   2.705  0.00834 **
## X5  -0.093432   0.108425  -0.862  0.39142
## X6  -0.063471   0.120010  -0.529  0.59835
## X7   0.035323   0.125730   0.281  0.77948
## X8  -0.042481   0.115028  -0.369  0.71287
## X9   0.085748   0.113017   0.759  0.45025
## X10  0.147267   0.119053   1.237  0.21971
## X11  0.043963   0.104601   0.420  0.67540
## X12 -0.037629   0.120575  -0.312  0.75579
## X13 -0.040844   0.121283  -0.337  0.73718
## X14  0.030316   0.126425   0.240  0.81110
## X15  0.239408   0.112427   2.129  0.03629 *
## X16  0.081389   0.130025   0.626  0.53313
## X17  0.171138   0.133239   1.284  0.20269
## X18  0.040340   0.108379   0.372  0.71072
## X19 -0.050519   0.110614  -0.457  0.64912
## X20  0.103824   0.108994   0.953  0.34368
## X21 -0.093361   0.127151  -0.734  0.46494
## X22 -0.062987   0.109931  -0.573  0.56827
## X23  0.119597   0.119399   1.002  0.31953
## X24  0.200846   0.125858   1.596  0.11447
## X25 -0.064935   0.115757  -0.561  0.57639
## X26  0.042791   0.105519   0.406  0.68617
## X27 -0.067227   0.107897  -0.623  0.53501
## X28 -0.037153   0.123721  -0.300  0.76473
## X29 -0.064788   0.127722  -0.507  0.61337
## X30 -0.142176   0.114476  -1.242  0.21788
## X31  0.187589   0.116862   1.605  0.11239
## X32 -0.101127   0.127979  -0.790  0.43175
## X33 -0.009285   0.110950  -0.084  0.93352
## X34 -0.135615   0.108108  -1.254  0.21333
## X35 -0.033501   0.105570  -0.317  0.75182
## X36  0.006990   0.116591   0.060  0.95234
## X37 -0.002173   0.105583  -0.021  0.98363
## X38 -0.164300   0.108265  -1.518  0.13306
## X39 -0.017016   0.109143  -0.156  0.87650
## X40 -0.033981   0.114801  -0.296  0.76800
## X41 -0.088039   0.118300  -0.744  0.45894
```

```
## X42  0.166009   0.119484   1.389  0.16857
## X43 -0.099728   0.114277  -0.873  0.38545
## X44  0.046778   0.114974   0.407  0.68520
## X45  0.139730   0.119479   1.169  0.24568
## X46  0.133005   0.110356   1.205  0.23167
## X47  0.082510   0.111679   0.739  0.46218
## X48  0.060008   0.119797   0.501  0.61781
## X49  0.071564   0.117202   0.611  0.54319
## X50  0.025570   0.114068   0.224  0.82320
## X51  0.017773   0.111185   0.160  0.87340
## X52 -0.012569   0.120808  -0.104  0.91740
## X53  0.127866   0.118374   1.080  0.28330
## X54 -0.063649   0.104397  -0.610  0.54380
## X55  0.029813   0.110874   0.269  0.78871
## X56 -0.069541   0.103458  -0.672  0.50341
## X57 -0.183767   0.124261  -1.479  0.14310
## X58  0.125368   0.115853   1.082  0.28245
## X59  0.013123   0.107267   0.122  0.90293
## X60 -0.049127   0.112818  -0.435  0.66441
## X61 -0.062615   0.101760  -0.615  0.54009
## X62 -0.049920   0.115007  -0.434  0.66541
## X63  0.073498   0.109782   0.669  0.50511
## X64  0.069620   0.100568   0.692  0.49077
## X65  0.014010   0.108928   0.129  0.89799
## X66 -0.023163   0.113457  -0.204  0.83875
## X67  0.001113   0.107764   0.010  0.99178
## X68 -0.253880   0.122114  -2.079  0.04082 *
## X69  0.023563   0.111688   0.211  0.83344
## X70 -0.012839   0.105635  -0.122  0.90357
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.001 on 80 degrees of freedom
## Multiple R-squared:  0.5047, Adjusted R-squared:  0.0713
## F-statistic: 1.165 on 70 and 80 DF,  p-value: 0.2541
```

Let's see how well the fitted values correspond to the true outcome values in training data and compare that to how well the predicted values in test data approximate the outcome values in test data.

```
fitted.tr = predict(lm.fit) # same as 'as.matrix(X.tr) %*% lm.fit$coeff'
fitted.te = predict(lm.fit, newdata = X.te) # same as 'as.matrix(X.te) %*% lm.fit$coeff'
par(mfcol = c(2,2))
par(mar = c(4, 4, 0.5, 0.5))
plot(fitted.tr, y.tr, xlab = "training fitted", ylab = "traininig y", pch = 19, col = "peru" )
plot(fitted.tr, y.tr - fitted.tr, xlab = "training fitted",
     ylab = "traininig y - fitted", pch = 19, col = "peru")
plot(fitted.te, y.te, xlab = "test fitted", ylab = "test y", pch = 19, col = "maroon1" )
plot(fitted.te, y.te - fitted.te, xlab = "test fitted",
     ylab = "test y - fitted", pch = 19, col = "maroon1")
```



Top row plots show how the fitted values from the model predict well in training data but not at all in test data. The residuals vs. fitted value plots (bottom row) show no pattern in training data, (which is good for a linear regression), but shows negative correlation in test data, which is to be expected when `y.te` and `fitted.te` are almost uncorelated.

Let's compute measures of fit by MSE and relate that to the total variance in the data (R2 value).

```
#MSE is the mean squared error of fitted values compared to observed values
#R2 is proportion of variance of y accounted by fitted values
data.frame(MSE = c(round(mean((y.tr - fitted.tr)^2),3),
                   round(mean((y.te - fitted.te)^2),3)),
           R2 = c(signif(1 - sum((y.tr - fitted.tr)^2)/sum((y.tr - mean(y.tr))^2),3),
                  signif(1 - sum((y.te - fitted.te)^2)/sum((y.te - mean(y.te))^2),3)),
           variance = c(round(var(y.tr),3), round(var(y.te),3)),
           row.names = c("Train","Test"))
```

```
##         MSE     R2 variance
## Train 0.534  0.504    1.083
## Test  1.596 -0.748    0.919
```

The model explains 50% of variance in training data, but R^2 estimate is negative in test data! This is because the MSE in test data is far larger than the variance of the test observations, meaning that by predicting each test data value by their common mean would give lower MSE than this model fit! This seems a completely useless model in the test data.

Not only has the model fit to the noise of training data but it also does not clearly pinpoint the real effects $x_1, \ldots, x_4$ that together would explain around 20% of y. We say that it has *overfit* to the training data. It has found artificial patterns between the outcome and predictors that are only statistical fluctuations and do not generalize to the other data sets. This happens when we have many possible predictors and a small sample size. The model is too flexible and fits to the noise. This is why we need to use shrinkage ideas, that do not just greedily try to explain the training data as well as possible, but will also penalize for more flexible models. Note that above the adjusted-R^2 as well as P-value based on F-statistic are saying that the model is not that good when we take into account the number of predictors used (also called degrees of freedom). But the standard R^2 or the model likelihood do not have any penalization for the model dimension and do not therefore tell about generalizability to other data sets.

Questions.

1. Linear model fit can be geometrically seen as a projection of n-dimensional outcome vector $\boldsymbol{y}$ on the subspace spanned by the columns of the predictor matrix $\boldsymbol{X}$ (see notes on Linear model). From this point of view, what happens to the model fit in the training data as we will add more (linearly independent) predictors in the model? How does overfitting look like in this geometric interpretation?
2. We always expect training error to be smaller than test error, so that property alone is not yet overfitting. What should be a definition for overfitting?

READ: *2.2.2 The Bias-Variance Trade-Off* from ISL.

- What is *Bias* and *Variance* of a statistical prediction method in intuitive terms?
- What is the formula that combines bias and variance to MSE?
- Which kinds of methods have high bias, which have high variance?
- Which kinds of methods overfit, which "underfit"?

Bias-variance trade-off

Let's decompose the expected test error into parts. Assume that we are looking at regression problem where $Y = f(X) + \varepsilon$ and $\mathrm{E}(\varepsilon) = 0$ and $\mathrm{Var}(\varepsilon) = \sigma^2$. We denote by $\widehat{f}$ the regression function that is learned on training data and we apply it to predict a new test observation whose predictors are $X = x_0$. We have

$$\begin{aligned}
\text{Test-err}(x_0) &= \mathrm{E}((Y - \widehat{f}(x_0))^2 \,|\, X = x_0) \\
&= \mathrm{E}((Y - f(x_0) + f(x_0) - \widehat{f}(x_0))^2 \,|\, X = x_0) \\
&= \mathrm{E}((Y - f(x_0))^2 \,|\, X = x_0) + \mathrm{E}((f(x_0) - \widehat{f}(x_0))^2) + 2 \cdot \mathrm{E}((Y - f(x_0))(f(x_0) - \widehat{f}(x_0))) \\
&= \sigma^2 + \mathrm{E}((f(x_0) - \mathrm{E}(\widehat{f}(x_0)) + \mathrm{E}(\widehat{f}(x_0)) - \widehat{f}(x_0))^2) + 2 \cdot \mathrm{E}(\varepsilon)\mathrm{E}(f(x_0) - \widehat{f}(x_0)) \\
&= \sigma^2 + \mathrm{E}((f(x_0) - \mathrm{E}(\widehat{f}(x_0)))^2 + \mathrm{E}(\mathrm{E}(\widehat{f}(x_0)) - \widehat{f}(x_0))^2 + 2 \cdot \mathrm{E}((f(x_0) - \mathrm{E}(\widehat{f}(x_0)))(\mathrm{E}(\widehat{f}(x_0)) - \widehat{f}(x_0))) + 2 \cdot 0 \\
&= \sigma^2 + \mathrm{Bias}^2(x_0) + \mathrm{Var}(\widehat{f}(x_0)) + 2 \cdot \mathrm{Bias}(x_0) \cdot 0 \\
&= \sigma^2 + \mathrm{Bias}^2(x_0) + \mathrm{Var}(\widehat{f}(x_0))
\end{aligned}$$

where $\mathrm{Bias}(x_0) = \mathrm{E}(\widehat{f}(x_0)) - f(x_0)$. (The covariance term with ε disappeared because error ε in test data has expectation 0 and is independent of both $f(x_0)$ and of $\widehat{f}(x_0)$ which is learned in the training data.) We see that the test MSE is bounded from below by *irreducible error* σ^2 that the regression model does not account for and therefore cannot decrease. We also see that to make test error as small as possible we should make both squared bias and variance small. However, these two terms are in inverse relation whence decrease of one leads to an increase of the other. In short, bias is increased when model is not flexible enough to model

the true f and variance is increased when the model is so flexible to fit data that small changes in data cause large variation in predicted $\widehat{f}$.

The problem of overfitting follows when we optimise the model fit (e.g. by maximising the likelihood function) but do not penalize for the number of parameters (or more generally, the flexibility of the model). By penalizing for the flexibility of the model we cause some bias, i.e., model is not completely free to go where (training) data points to but instead we prefer sparser models that hopefully are better generalizable to test data. This bias means that we will not be able to fit perfectly to settings where best models are not sparse. However, by inducing this bias, we also decrease variance of the model fit, because it is not anymore free to follow the training data and therefore it is more robust to changes in data and, in particular, is less prone to overfit to training data. In high-dimensional problems it is crucial to look for an approriate tradeoff between bias and variance to keep the model flexible enough to truly learn from the data (which requires lowering bias) while at the same time minimizing overfitting (which requires lowering variance).

Questions. (From ISL)

1. For each of parts (a) through (d), indicate whether we would generally expect the performance of a flexible statistical learning method to be better or worse than an inflexible method. Justify your answer.

(a) The sample size n is extremely large, and the number of predictors p is small.
(b) The number of predictors p is extremely large, and the number of observations n is small.
(c) The relationship between the predictors and response is highly non-linear.
(d) The variance of the error terms, i.e. $\sigma^2 = \text{Var}(\varepsilon)$, is extremely high.

2. We now revisit the bias-variance decomposition. Provide a sketch of typical (squared) bias, variance, training error, test error, and irreducible error curves, on a single plot, as we go from less flexible statistical learning methods towards more flexible approaches. The x-axis should represent the amount of flexibility in the method, and the y-axis should represent the values for each curve. There should be five curves. Explain why each of the five curves has the shape displayed.

How to choose suitable models: Information criteria

As we saw, the model containing all of the predictors in training data will always have the smallest RSS and the largest R2, since these quantities are related to the training error. Instead, we wish to choose a model with a low test error. The training error can be a poor estimate of the test error. Therefore, RSS and R2 are not suitable for selecting the best model among a collection of models with different numbers of predictors. Let's have a look at two commonly used information criteria that attempt to better estimate model's predictive ability also outside the training data, but still use only training data as input.

Akaike Information Criterion (AIC)

AIC is a general quantity to compare different models defined on the same data set. It is named according to Hirotugu Akaike (published early 1970s).

Suppose we consider models $M_0, \ldots, M_J$ where model M_j has k_j parameters. Let $L_j(\widehat{\theta}_j)$ be the maximum value of the likelihood function for the model M_j. Then the AIC value of model M_j is

$$\text{AIC}_j = 2k_j - 2\log(L_j(\widehat{\theta}_j))$$

The smaller the AIC, the better the model (as measured by AIC).

We know that by increasing the model complexity (e.g. number of parameters) we can make model fit better to data and hence get higher values of likelihood. AIC penalizes the maximized log-likelihood by the number of parameters of the model. Thus, if two models have quite similar maximized log-likelihoods, then the one with less parameters is preferred. Intuitively this favors sparser models and therefore helps to decrease overfitting.

AIC is derived by an information theoretic argument. Suppose that the data are generated by a probability density f. Consider two candidate densities g_1 and g_2 to represent f. If we knew f, then the information lost when using g_1 to represent f is quantified by the Kullback-Leibler divergence,

$$\mathrm{D_{KL}}(f \,||\, g_1) = \int f(x) \log\left(\frac{f(x)}{g_1(x)}\right) \mathrm{d}x.$$

Similarly, we could compute $\mathrm{D_{KL}}(f \,||\, g_2)$ and then choose the candidate model that minimized the expected information loss.

In practice, we of course do not know f. Akaike (1974) showed, however, that we can estimate, via differences in AIC, how much more (or less) information is lost by g_1 than by g_2. This estimate, though, is only valid asymptotically and makes many quite crude approximations. In particular, if the number of data points is small, then some correction is often necessary (see e.g. AICc).

Connection to likelihood ratio test in nested models. If model M_1 is a submodel of model M_2 (e.g. both are regression models but in M_2 we use $k_2 - k_1$ more predictors on top of what is used in M_1), then, if the simpler model M_1 holds, Wilks thorem says that the likelihood ratio test statistic

$$\mathrm{LRT}_{12} = -2 \log\left(\frac{L_1(\widehat{\theta}_1)}{L_2(\widehat{\theta}_2)}\right) \sim \chi^2_{k_2-k_1},$$

which can be used to derive P-values to test the additional parameters. (This result is asymptotic and requires some regularity conditions and, for example, that the additional parameters of model M_2 are not at boundary values of their range in M_1.) We see that the AIC difference can be written in terms of LRT as

$$\Delta_{12} = \mathrm{AIC}_1 - \mathrm{AIC}_2 = 2(k_1 - k_2) + \mathrm{LRT}_{12}.$$

Given that $\mathrm{E}\left(\chi^2_{k_2-k_1}\right) = k_2 - k_1$, the expectation of Δ_{12} is

$$\mathrm{E}(\Delta_{12} \,|\, M_1) = 2(k_1 - k_2) + (k_2 - k_1) = k_1 - k_2 < 0$$

showing that AIC is expected to correctly favor the model M_1.

Note that AIC can be compared for any pair of models on the same data, whereas the $\chi^2_{k_2-k_1}$ distribution for LRT holds only for *nested* models.

Bayesian Information Criterion (BIC)

BIC was introduced by Gideon E Schwarz in 1978, and is also called as Schwarz criterion. BIC has a similar form to AIC, but imposes a stronger penalty on the number of parameters. With the same notation as was used above for AIC, we define BIC value of model M_j as

$$\mathrm{BIC}_j = \log(n) k_j - 2 \log(L_j(\widehat{\theta}_j)),$$

where n is the number of (indepedent) data points, often the sample size.

The BIC is derived using a Bayesian argument assuming that the prior distribution of the parameters is flat, i.e. a constant, in the region of interest. Then we have a following approximation for the marginal likelihood by using Laplace approximation to compute the integral:

$$\Pr(y \,|\, M_j) = \int \Pr(\theta_j \,|\, M_j) \Pr(y \,|\, \theta_j, M_j) \mathrm{d}\theta_j \approx C\, n^{-k_j/2}\, L_j(\widehat{\theta}_j),$$

where C is a constant not depending on n. Thus we have the approximation

$$2 \log(\Pr(y \,|\, M_j)) \approx 2 \log\left(L_j(\widehat{\theta}_j)\right) - k_j \log(n) + C' = -\mathrm{BIC}_j + C'$$

Hence BIC can be seen as an approximation to (negative) log-marginal likelihood of the model and smaller BIC corresponds to larger marginal likelihood, and hence better explanation of the data, *after* the model complexity has been taken into account.

By ignoring the constants, we have an approximation to the Bayes factor between models 1 and 2 as

$$\log(\text{BF}_{12}) = \log\left(\frac{\Pr(y \mid M_1)}{\Pr(y \mid M_2)}\right) \approx \frac{1}{2}(\text{BIC}_2 - \text{BIC}_1).$$

This gives an interpretation for comparing models via difference in BIC values.

In particular, note how Bayesian marginal likelihood automatically accounts for the number of parameters of the model, and can get higher *or lower* as we move to more complex models (see "The role of marginal likelihood in Bayesian inference" from lecture HDS4). This is in contrast with the maximum likelihood value that can only get higher as we increase the number of parameters of the model. The difference between marginal likelihood and maximum likelihood is that the former has been computed by averaging the likelihood function with respect to the prior distribution of the parameters, whereas the latter only maximizes the likelihood and ignores the prior distribution. In BIC, the approximation that takes us from the maximized likelihood to an approximate marginal likelihood is the addition of the penalty term $\log(n)k$, where k is the number of parameters.

For a detailed derivation of BIC see, for example, https://www.researchgate.net/publication/265739543_On_the_Derivation_of_the_Bayesian_Information_Criterion. And for detailed discussion about both AIC and BIC http://mathfaculty.fullerton.edu/sbehseta/AIC.vs.BIC.pdf.

Example 5.2. We can determine the P-value thresholds at which AIC and BIC start to favor the more complex model H_1 (with maximized likelihood L_1) over the simpler submodel H_0 (with maxmized likelihood L_0), when the difference in parameters is k and the models are nested (such that LRT is applicable).

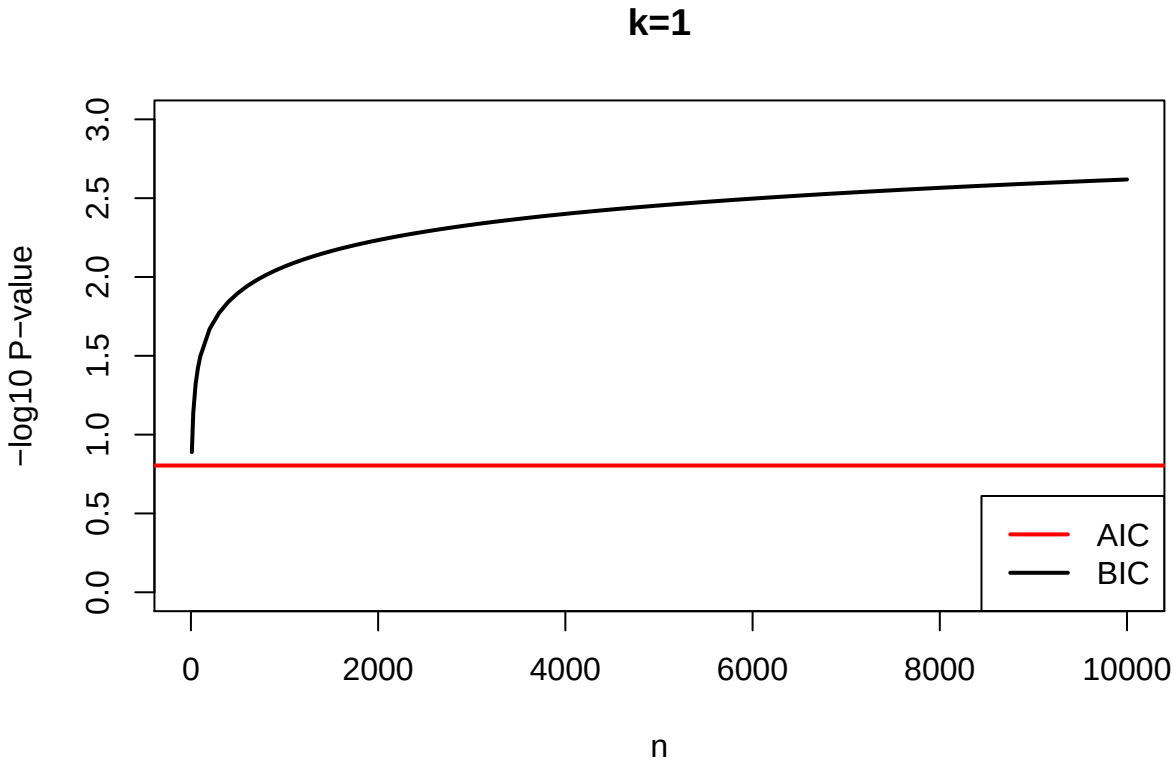
$$\text{AIC}_0 > \text{AIC}_1 \iff -2\log(L_0) > 2k - 2\log(L_1) \iff \text{LRT}_{01} > 2k \iff \text{P-value} < 1 - F_{\chi^2_k}(2k)$$

and for $k = 1$ this threshold is 0.157.

$$\text{BIC}_0 > \text{BIC}_1 \iff -2\log(L_0) > k\log(n) - 2\log(L_1) \iff \text{LRT}_{01} > k\log(n) \iff \text{P-value} < 1 - F_{\chi^2_k}(k\log(n))$$

This threshold depends on n as follows:

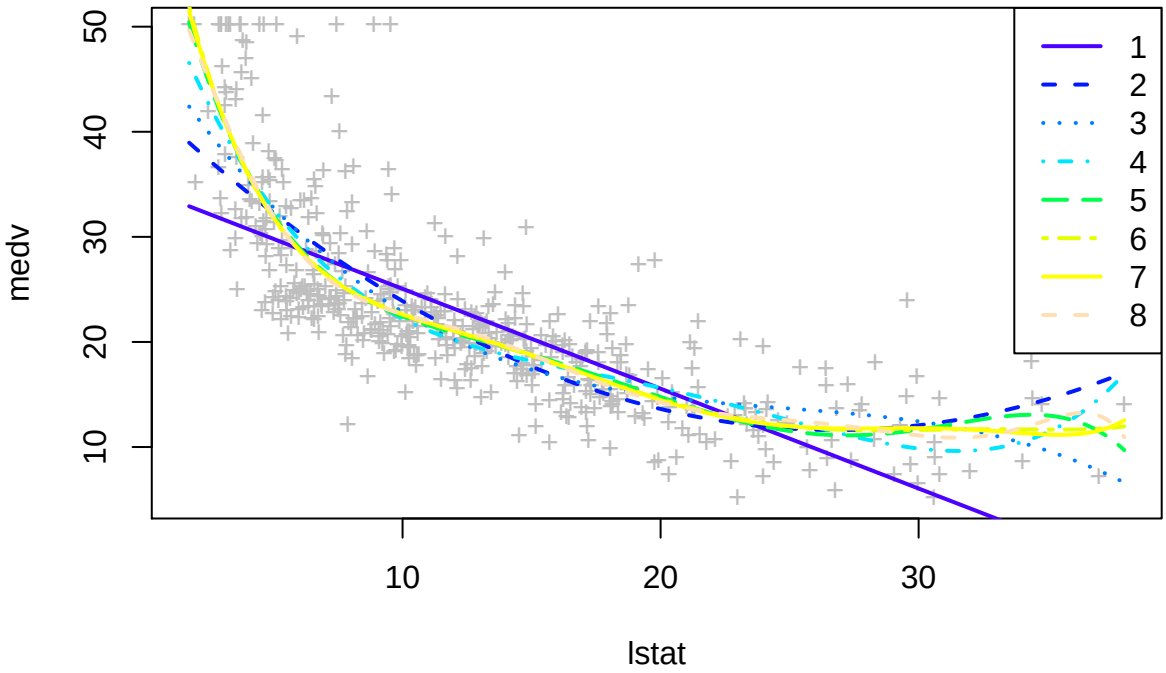
```
k = 1
ns = c(10, 25, 50, 75, seq(100,10000,100))
plot(ns, -log10(pchisq(k*log(ns), df = 1, lower = F)),
     t = "l", lwd = 2, ylab = "-log10 P-value", xlab = "n", ylim = c(0,3), main = "k=1") #BIC threshold
abline(h = -log10(0.157), col = "red", lwd = 2) #AIC threshold
legend("bottomright", leg = c("AIC", "BIC"), col = c("red", "black"), lwd = 2)
```



This shows that AIC is much more liberal than BIC in letting additional variables in the model and the difference increases as information in data (n) grows.

Example 5.3. Let's see how AIC and BIC work when we fit different polynomial models for `Boston` data set trying to predict `medv` (median value of a property) with `lstat` (proportion of lower status people in area).

```
library(MASS)
medv = Boston$medv
lstat = Boston$lstat
plot(lstat, medv, pch = "+", col = "gray")
x.points = seq(min(lstat), max(lstat), length = 1000) #grid on which to evaluate
cols = topo.colors(8)
Bos.res = data.frame(matrix(NA, ncol = 2, nrow = 8))
names(Bos.res) = c("AIC","BIC")
for(ii in 1:8){
  Bos.fit = lm(medv ~ poly(lstat, ii, raw = T), data = Boston) #ii deg polynomial
  y.at.x = predict(Bos.fit, data.frame(lstat = x.points))
  lines(x.points, y.at.x, col = cols[ii], lty = ii, lwd = 2)
  Bos.res[ii, ] = c(AIC(Bos.fit), BIC(Bos.fit))
}
legend("topright", leg = 1:8, col = cols, lwd = 2, lty = 1:8)
```




```
Bos.res
```

```
##        AIC      BIC
## 1 3288.975 3301.655
## 2 3170.516 3187.422
## 3 3147.796 3168.928
## 4 3126.856 3152.216
## 5 3115.247 3144.833
## 6 3115.668 3149.481
## 7 3117.528 3155.566
## 8 3117.834 3160.099
```

We see that both AIC and BIC are minimized by the 5th degree polynomial, so this would be the best model according to these criteria. This would be the bias-variance compromise here: The smaller degree polynomials don't explain data well enough and higher degree polynomials start to overfit. We could also check whether 5th degree polynomial gives the smallest test error in a test set, but for that we would need to first split our data into test and training sets. (We will later look how to do this with cross-validation.)

READ: 6.1.1-6.1.2 *Subset selection* from ISL.

- What are problems with best subset selection?
- What are advantages of stepwise selection methods?

Stepwise regression

By using, for example, either of AIC or BIC we could compare and rank different regression models even with different numbers of parameters. When there are p predictors, there are 2^p different subset models, and *best subset selection* becomes impossible already when p gets to the range of 20-30. Instead, a simple way to search for a top model is through a stepwise procedure where we start from the empty model, we evaluate putative additions of all possible predictors that are not yet in the model and choose the one which gives the optimal value for the criterion. We continue this until no new predictor increases the criterion. This is called *forward selection*. We can also start from the full model and by *backward selection* start dropping terms that

are not needed. We may also add a possibility at each step to either drop existing predictors or add new ones in *hybrid selection*. Note that in both forward and backward selection there are at most p steps and hence at most $\frac{1}{2}(p^2+p)$ models to fit, compared to 2^p models in best subset selection.

Let's see how to do this in R. We have our regression model in `lm.fit` where we had fit $p = 70$ predictors of which only the first four truly affected the outcome. We don't plot the output of these functions in whole here, but you can see it yourself by removing `trace=0` from the code blocks.

```
aic.back = step(lm.fit, direction = "backward", k = 2, trace = 0)
#k = 2 corresponds to AIC i.e. penalty is 2*#params
#which variables are in the final model chosen by AIC
#and what are their coefficients in that model
aic.back.coeff = coefficients(aic.back)
aic.back.vars = names(aic.back.coeff)
aic.back.vars
```

```
##  [1] "X1"  "X2"  "X3"  "X4"  "X15" "X17" "X21" "X24" "X31" "X34" "X38"
## [12] "X42" "X45" "X46" "X58" "X68"
```

```
#we get BIC by setting k = log(n) where n is the number of individuals in regression
bic.back = step(lm.fit, direction = "backward", k = log(n.tr), trace = 0)
bic.back.coeff = coefficients(bic.back)
bic.back.vars = names(bic.back.coeff)
bic.back.vars
```

```
## [1] "X1" "X2" "X3" "X4"
```

So we see that BIC is much more stringent and keeps only 4 variables whereas AIC keeps 16. Both include the true effects and BIC finds exactly the true model.

We have the test set, so let's see how these two models work in out-of-sample data, which should be our ultimate goal to judge what is a good model.

```
paste("Test MSE for AIC back",signif(mean((y.te - as.matrix( X.te[,aic.back.vars]) %*% aic.back.coeff)^
```

```
## [1] "Test MSE for AIC back 1.3"
```

```
paste("Test MSE for BIC back",signif(mean((y.te - as.matrix(X.te[,bic.back.vars]) %*% bic.back.coeff)^2
```

```
## [1] "Test MSE for BIC back 0.82"
```

So BIC is better, as expected as it is using exactly the correct variables. Note that the variance in test data was about 0.92, so the variance explained by BIC model here is $1 - 0.82/0.92 = 10.8\%$. (We would expect close to 20% variance explained if the coefficients were accurately estimated, but here the sample size is too small for very accurate estimates.)

Let's do forward selection instead. Here we need to specify the starting model and the largest possible model.

```
aic.fwd = step(lm(y.tr ~ 0, data = X.tr), scope = formula(lm(y.tr ~ 0 + . , data = X.tr)),
               direction = "forward", k = 2, trace = 0)
aic.fwd.coeff = coefficients(aic.fwd)
aic.fwd.vars = names(aic.fwd.coeff)
aic.fwd.vars
```

```
##  [1] "X1"  "X3"  "X4"  "X2"  "X10" "X34" "X64" "X38" "X46" "X45" "X26"
```

```
bic.fwd = step(lm(y.tr ~ 0, data = X.tr), scope = formula(lm(y.tr ~ 0 + . , data = X.tr)),
               direction = "forward", k = log(n.tr), trace = 0);
bic.fwd.coeff = coefficients(bic.fwd)
bic.fwd.vars = names(bic.fwd.coeff)
bic.fwd.vars
```

```
## [1] "X1"  "X3"  "X4"  "X2"  "X10"
```

Now, on top of the 4 correct variables, BIC gave one additional variable and AIC gave 7 variables. Let's test these models in test data.

```
paste("Test MSE for AIC fwd",signif(mean((y.te - as.matrix( X.te[,aic.fwd.vars]) %*% aic.fwd.coeff)^2),1
```

```
## [1] "Test MSE for AIC fwd 1"
```

```
paste("Test MSE for BIC fwd",signif(mean((y.te - as.matrix(X.te[,bic.fwd.vars]) %*% bic.fwd.coeff)^2),2)
```

```
## [1] "Test MSE for BIC fwd 0.86"
```

For BIC, fwd search gave a slight worse model than backward search whereas the opposite is true for AIC.

Our example here was about regression where the true model was very sparse (only 4/70 predictors were relevant). Here BIC worked much better than AIC, because BIC leads to sparser models in general. In cases when there are more relevant predictors, each with smaller effect size, the picture may be different.

In general, there is no clear choice between AIC and BIC. Theoretically, BIC is asymptotically consistent as a selection criterion. This means that given a family of models, including the true model, the probability that BIC will select the correct model approaches one as the sample size $n \to \infty$. This is not the case for AIC, which tends to choose models which are too complex as $n \to \infty$. However, this results assumes that the true model is among the tested ones, which is unlikely to be the case in real world applications. In practice, AIC is preferred in situations when a false negative finding would be considered more misleading than a false positive, and BIC is preferred in situations where a false positive is as misleading as, or more misleading than, a false negative.

Let's have a look at the Boston data set.

Example 5.4. Let's predict `chas` that is a Binary variable using logistic regression. (With logistic regression we should always include the intercept.) Let's do forward and backward regressions.

```
library(MASS)
#Forward AIC
step(glm(chas ~ 1, family = "binomial", data = Boston),
     scope = formula(glm(chas ~ ., family = "binomial", data = Boston)),
     trace = 0, k = 2, direction = "forward")
```

```
##
## Call:  glm(formula = chas ~ medv + nox + crim + indus, family = "binomial",
##     data = Boston)
##
## Coefficients:
## (Intercept)         medv          nox         crim        indus
##    -7.72390      0.07704      5.04992     -0.13759      0.05504
##
## Degrees of Freedom: 505 Total (i.e. Null);  501 Residual
## Null Deviance:       254.5
## Residual Deviance: 223   AIC: 233
```

```
step(glm(chas ~ ., family = "binomial", data = Boston), trace = 0, k = 2, direction = "backward")
```

```
##
## Call:  glm(formula = chas ~ crim + indus + nox + rad + tax + medv, family = "binomial",
##     data = Boston)
##
## Coefficients:
## (Intercept)         crim        indus          nox          rad
##   -6.574906    -0.239016     0.095828     6.064839     0.189533
```

```
##         tax         medv
##   -0.009217     0.071458
##
## Degrees of Freedom: 505 Total (i.e. Null);  499 Residual
## Null Deviance:       254.5
## Residual Deviance: 214.5     AIC: 228.5
```

Backward selection found a model with lower AIC than forward selection. Why is this? The top model has variables `rad` and `tax` on top of what the forward selection model found. Let's see what happens when we put these two variables in the model separately vs together.

```
AIC(glm(chas ~ medv + nox + crim + indus, family = "binomial", data = Boston)) #AIC-fwd model
```

```
## [1] 232.9887
```

```
AIC(glm(chas ~ medv + nox + crim + indus + rad, family = "binomial", data = Boston)) # add only rad
```

```
## [1] 233.6105
```

```
AIC(glm(chas ~ medv + nox + crim + indus + tax, family = "binomial", data = Boston)) # add only tax
```

```
## [1] 234.2564
```

```
AIC(glm(chas ~ medv + nox + crim + indus + rad + tax, family = "binomial", data = Boston)) #add rad and
```

```
## [1] 228.4741
```

We see that adding either `tax` or `rad` alone would not decrease AIC but adding them together would decrease AIC. This shows that stepwise search may not find the optimal model because a single step may not always be enough to get from the current model to a region of better models, even when such a region existed. Thus running both forward and backward search might be useful, although does not guarantee that we would find the optimal model either.

What about adding interaction terms? Compact notation `.^2` means all main terms and all pairwise interaction terms.

```
aic.interaction = step(glm(chas ~ 1, family = "binomial", data = Boston),
                       scope = formula(glm(chas ~ .^2, family = "binomial", data = Boston)),
                       trace=0, k = 2, direction = "forward")
aic.interaction
```

```
##
## Call:  glm(formula = chas ~ medv + nox + crim + age + indus + rad +
##     tax + dis + ptratio + rm + black + medv:crim + nox:crim +
##     nox:age + medv:nox + medv:indus + medv:age + nox:tax + crim:indus +
##     indus:ptratio + age:rad + medv:dis + crim:rm + tax:dis +
##     nox:dis + nox:ptratio + nox:indus + nox:rm + indus:rm, family = "binomial",
##     data = Boston)
##
## Coefficients:
##   (Intercept)           medv            nox           crim            age
##    -41.116893       1.307627      48.797826     -13.354216       1.055293
##         indus            rad            tax            dis        ptratio
##      1.119478      -0.317790      -0.036844     -20.258624       7.733194
##            rm          black      medv:crim       nox:crim        nox:age
##    -10.678576       0.023318       0.157145      36.870596      -1.696838
##      medv:nox     medv:indus       medv:age        nox:tax     crim:indus
##     -5.271692       0.117138      -0.009986       0.179881      -1.305001
## indus:ptratio        age:rad       medv:dis        crim:rm        tax:dis
```

```
##     -0.121755       0.016140       0.255298       0.954968      -0.029716
##       nox:dis    nox:ptratio      nox:indus         nox:rm       indus:rm
##     42.764383     -15.545598       4.070428      27.408452      -0.495798
##
## Degrees of Freedom: 505 Total (i.e. Null);  476 Residual
## Null Deviance:       254.5
## Residual Deviance: 98.23     AIC: 158.2
```

Why it might not be a good idea to do backward search from model with all interaction terms in it?

For the chosen interaction model we see a lot lower AIC than with the main effects. Can we see whether this model truly performs well and does not just overfit to the data? That is a bit late now, since the model has been chosen in the whole data set. Even if we re-estimated the effect sizes of the predictors and interactions in only some subset of the data and applied it to another subset, we would still underestimate the true test error that we would expect in an independent test data set.

Question: How could we compare performance of the models when we **do not have an extra test data set** and do not want to split our valuable data for a separate test set, which would decrease our power to estimate a good model because the sample size in training data would be smaller after a test data set has been removed?

READ: 5.1.1-5.1.4 *Cross-validation* from ISL.

- What does cross-validation attempt to do?
- What is LOOCV and K-fold CV and how do they compare?

Cross-validation

The idea of cross-validation (CV) is to mimic the split of data into training and validation sets, without completely ignoring any part of the data in the training part. (For us here "validation set" is a synonym for "test set".) In K-fold CV, we split the data into K parts, and carry out the following procedure for each part $j = 1, \ldots, K$:

- Use other parts of the data except part j to train the model.
- Validate the trained model on part j and record the test measure (such as MSE for continuous variables or correct classification for logistic regression model).

After all K parts have been used as a validation set

- Average the test measure over all K runs.

By carrying CV out for different models, we approximate each model's performance on out-of-sample test data. Typically $K = 5$ or $K = 10$ are used and by keeping K rather small we avoid large variance due to small validation sets, but still have multiple parts in order to avoid the bias present in any one split of the data set. The case $K = n$ is known as leave-one-out cross-validation. In this case, for observation i, the fit is computed using all the data except i.

Let's try cross-validation on our original data with $n = 150$ training samples and $p = 70$ predictors. The goal is to estimate test error of the model that includes all variables by using just the training data set. CV is implemented in `cv.glm()` function of the `boot` package and requires model fitted by `glm()`, that by default fits a linear model with Gaussian errors. Thus, by default, `glm()` fits the same model as `lm()` but the output is a bit different as `glm()` is also applicable to other regression models such as logistic regression (using `glm(,family="binomial")`).

```
library(boot) #has cv.glm function to do CV for (generalized) linear models
glm.fit = glm(y.tr ~ 0 + ., data = X.tr)  #linear model w/o intercept fitted by glm
cv.glm( data.frame(y.tr, X.tr), glm.fit, K = 10)$delta #K=10 fold CV using MSE as cost function
```

```
## [1] 1.999163 1.896178
```

```
#outputs 2 values: 1 = raw CV estimate and 2 = adjusted CV estimate of prediction error.
var(y.tr) #for comparison with CV MSE.
```

```
## [1] 1.083105
```

We estimate a test MSE of about 2.0 which is nearly 100% larger than the variance in training data y. CV is telling that it is not a good idea to fit this kind of linear model ($p = 70$) with this small sample size ($n = 150$) as the estimated MSE is much larger than variance of the outcome variable. In other words, the model is useless when applied to new data.

Input for `cv.glm()` can also include a cost function of two vector arguments. The first argument to cost should correspond to the observed responses and the second argument should correspond to the predicted or fitted responses from the generalized linear model. The default is MSE, and it is used here.

The Wrong and Right Way to Do Cross-validation (from ESL 7.10.2)

Consider a problem with a large number of predictors. A typical strategy for analysis might be as follows:

1. Screen the predictors: find a subset of "good"" predictors that show fairly strong (univariate) correlation with the outcome.
2. Using just this subset of predictors, build a multivariate model.
3. Use cross-validation to estimate the unknown tuning parameters and to estimate the prediction error of the final model.

Is this a correct application of cross-validation?

The problem is that the predictors have an unfair advantage, as they were chosen in step (1) on the basis of all of the samples. Leaving samples out after the variables have been selected does not correctly mimic the application of the model to a completely independent test set, since these predictors "have already seen" the left out samples.

Here is the correct way to carry out cross-validation in this example:

1. Divide the samples into K cross-validation folds at random.
2. For each fold $k = 1, 2, \ldots, K$

- Find a subset of "good" predictors that show fairly strong (univariate) correlation with the outcome, using all of the samples except those in fold k.
- Using just this subset of predictors, build a multivariate model, using all of the samples except those in fold k.
- Use the model to predict the outcome for the samples in fold k.

The error estimates from step 2(c) are then accumulated over all K folds, to produce the cross-validation estimate of prediction error. In general, with a multistep modeling procedure, cross-validation must be applied to the entire sequence of modeling steps. In particular, validation samples must be "left out" before any selection or filtering steps that use outcome variable are applied.

Using observed data for deriving sampling distributions

Cross-validation is an example of *resampling methods* where we use our observed data to not only fit some model, but also evaluate its performance. When we have a large number of observations from the target population, then several other methods with similar spirit are also very useful.

When we considered Q-values, we already saw how the observed P-value distribution can be used to infer π_0, the proportion of true nulls, and further to approximate local false discovery rate for any one P-value.

Two other common applications are permutation testing and Bootstrap.

Note the difference between these methods and the traditional way of stipulating a theoretical distribution for a test statistic and then comparing the observed to the theoretical.

Permutation testing

In permutation testing we permute the indexes of sets of variables with respect to each other in order to break all associations *between* the sets while maintaining all the structure *within* the sets. For example, suppose that I want to fit a non-linear spline model to predict disease risk (y) from set of 20 clinically measured variables (X). We don't have a reliable theoretical null distribution to know what kind of MSE this complex and highly non linear prediction method would give already when there is no association between y and X. In permutation testing, we would permute the elements of y vector and run the method on permuted data exactly as it was run on the original unpermuted data. By collecting MSE values from 1000s of permuted data sets we would have an idea of the null distribution of MSE from this method. Importantly, we would have realistic structure among the predictors X, since the columns of X were not permuted. We only permuted away the association between y and X, not between any two columns of X.

Bootstrap

We are not considering bootstrap on this course. It is explained in *5.2 The Bootstrap* in ISL.