Chapter 5. Overfitting and Its Avoidance

"*If you torture the data long enough, it will confess*" – Nobel Laureate Ronald Coase

- *Basic concepts*: Fitting and overfitting, complexity control, generalization
- *Exemplary techniques*: Cross-validation, variable selection, tree pruning, regularization

Further readings:

James et al. (2013) Sections 5.1, 5.3.1-5.3.3, 6.1-6.2 & 6.5-6.6,

Provost and Fawcett (2013) Chapter 5.

If we allows ourselves enough flexibility in searching for patterns in a data set, we will find pattern. Unfortunately those ‘patterns’ may be just chance occurrences in the data. They do not represent systematic characteristics of the underlying system.

Overfitting: a model is tailored too much to the training data at the expense of generalization to previously unseen data point.

Overfitting is associated with model complexity: more complex a model is, more likely it is overfitting.

A extreme case of overfitting is the so-called *table model* which records all the training data exactly. A table model performs typically poorly in *generalization*, i.e. it often predict a new value badly.

A tree table model: Grow tree by keeping splitting on variables until there is a single point at each leaf node.

A regression table model: Fitting $y = \beta_0 + \beta_1 x + \cdots + \beta_{n-1} x^{n-1}$ with to the data $(y_i, x_i), i = 1, \cdots, n$ results zero residuals.

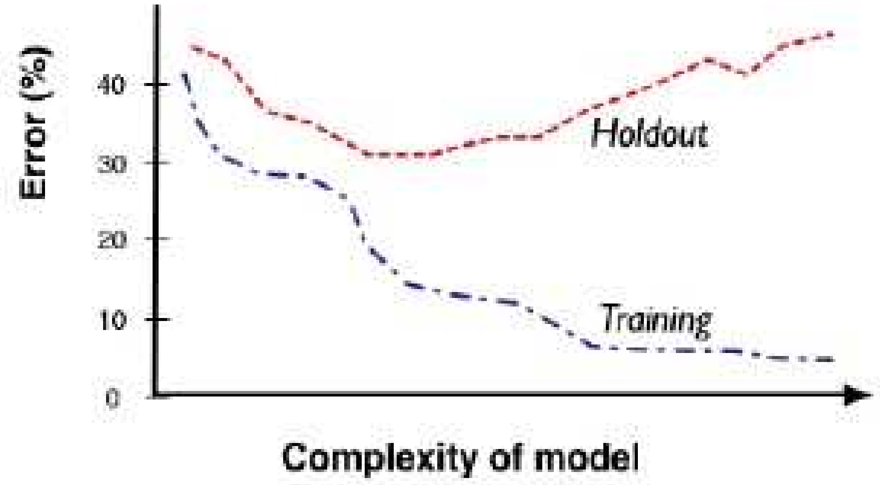
In general data are regarded containing signal with noise. A good model should catch signal but leave out the noise. An overfitting takes also noise as a part of signal.

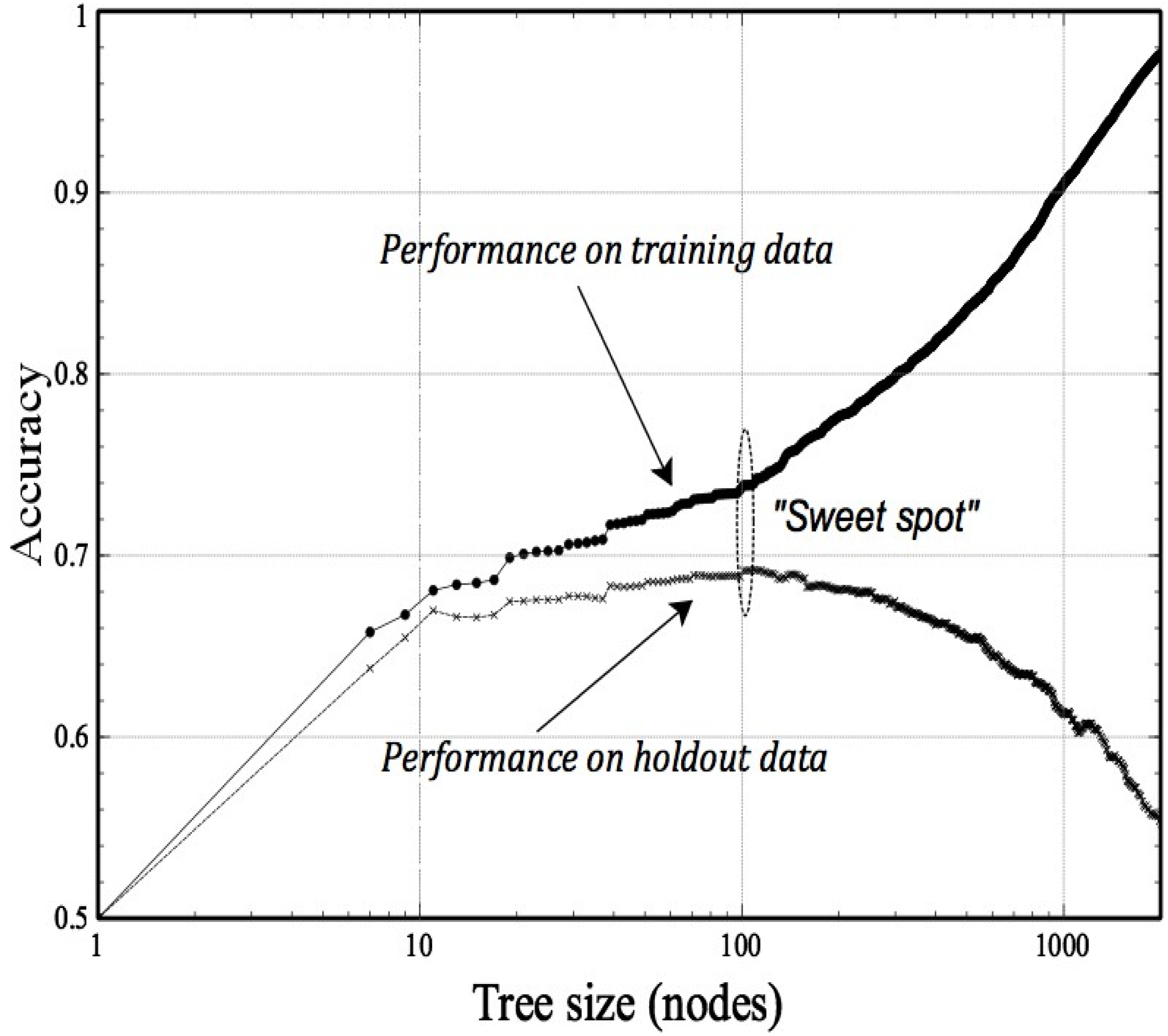
Difficulty: how much noise in data is unknown.

Check for overfitting: check the performance of a fitted model on the data not used in fitting.

- Learning (training) data: data used in fitting

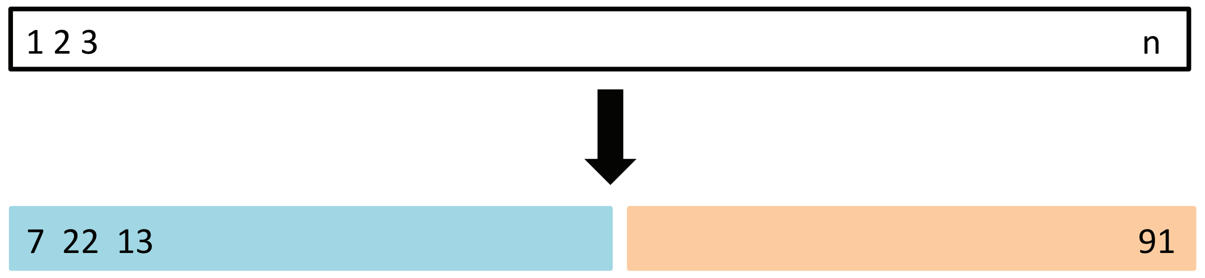
- Validation (holdout) data: data not used in fitting, held for testing the performance of a fitted model


Error (%)
40
30
20
10
0
Holdout
Training
Complexity of model



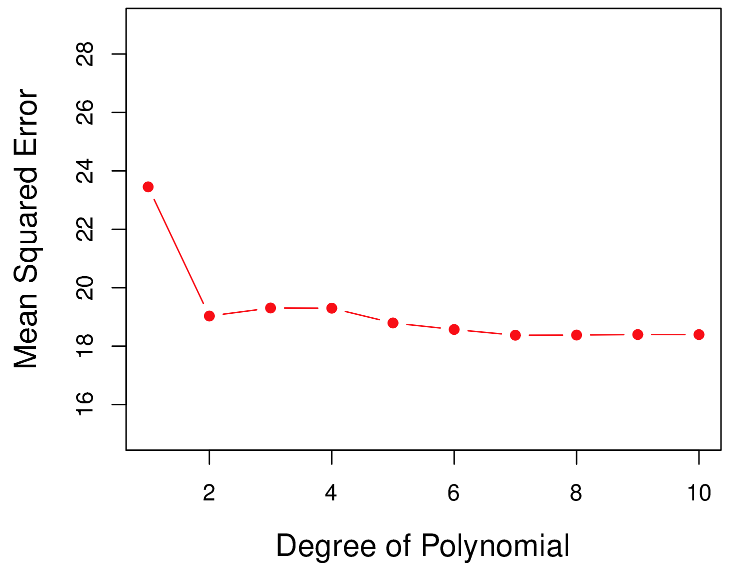
1
0.9
0.8
0.7
0.6
0.5
Accuracy
1
10
100
1000
Tree size (nodes)
Performance on training data
"Sweet spot"
Performance on holdout data

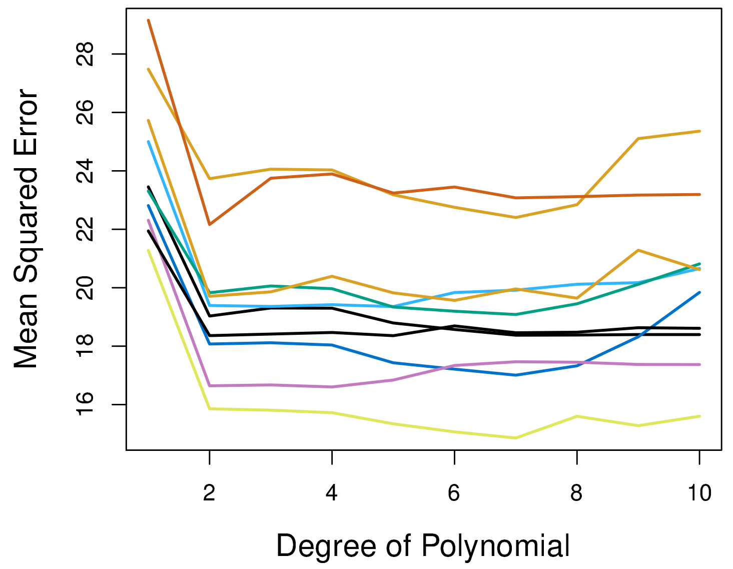
Holdout Evaluation. Divide the available data into two parts: training data used to fit a model, and holdout data for validation.



Two drawbacks:

- The usage of the data is not efficient: both training and validation data sets are smaller than the available data.
- The validation error depends on the particular split of validation set and training set.





The holdout evaluation is applied to `Auto.txt` with models

$$\texttt{mpg} = \beta_0 + \beta_1 \texttt{hpower} + \cdots + \beta_p \,\texttt{hpower}^p + \varepsilon$$

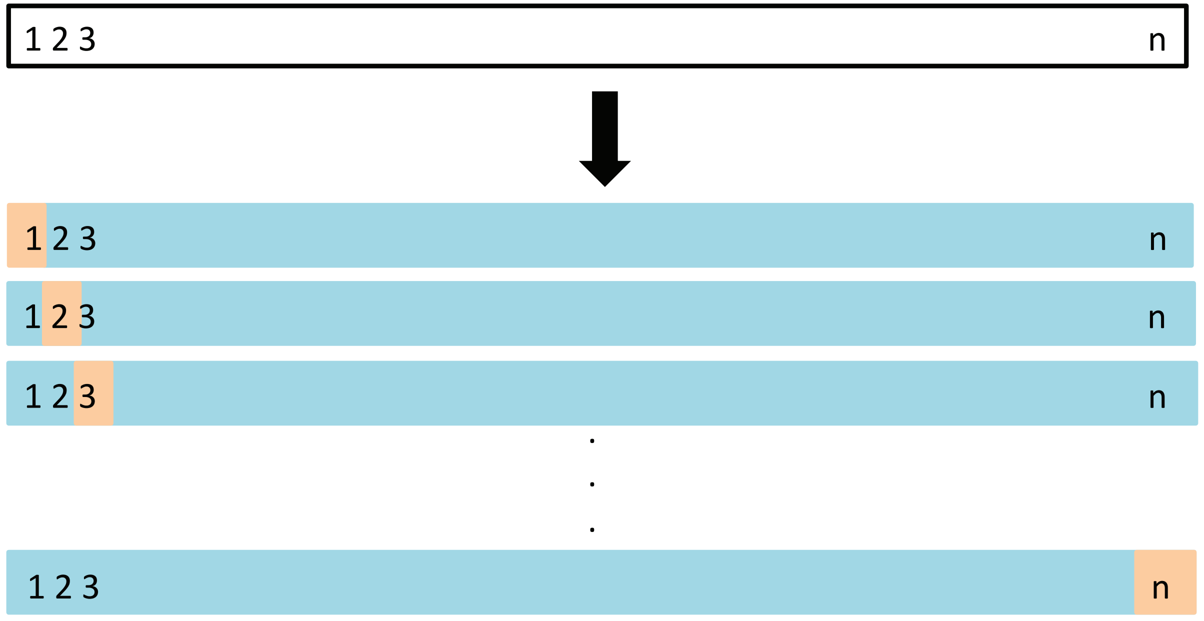
for $p = 1, \cdots, 10$. MSE is the average $(\widehat{y} - y)^2$ for y over a holdout set, and $\widehat{y}$ is the predicted value of y based on the model estimated using the training set.

Left panel: Plot of MSE vs p for one split between training and holdout sets.

Right panel: for 10 different splits.

Cross-validation (CV) – choosing model complexity by minimizing validation errors

Leave-one-out approach: each time leave one data point out, fit the model using all the other data, calculate the error on the point left out. Repeat this process for each data point. The best model should minimize the accumulative errors.



Computationally intensive, but more efficient use of the data. The MSE is calculated as

$$CV(w) = \frac{1}{n}\sum_{j=1}^{n}(y_j - \widehat{y}_j(w))^2$$

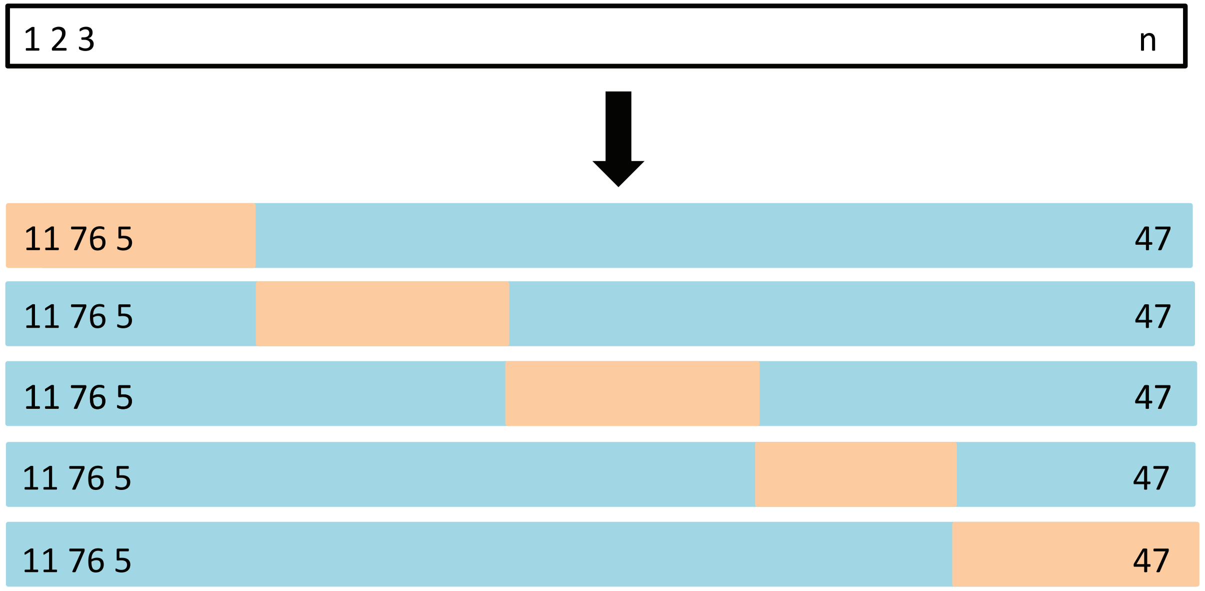
where $\widehat{y}_j$ is the predicted value for y_j based on the estimated model with the complexity indexed by w using the other $(n-1)$ observations.

The CV selected model should have the complexity $\widehat{w}$ which minimizes $CV(w)$.

Complexity w: the number of regressors in a regression model, or the number of terminal nodes of a tree.

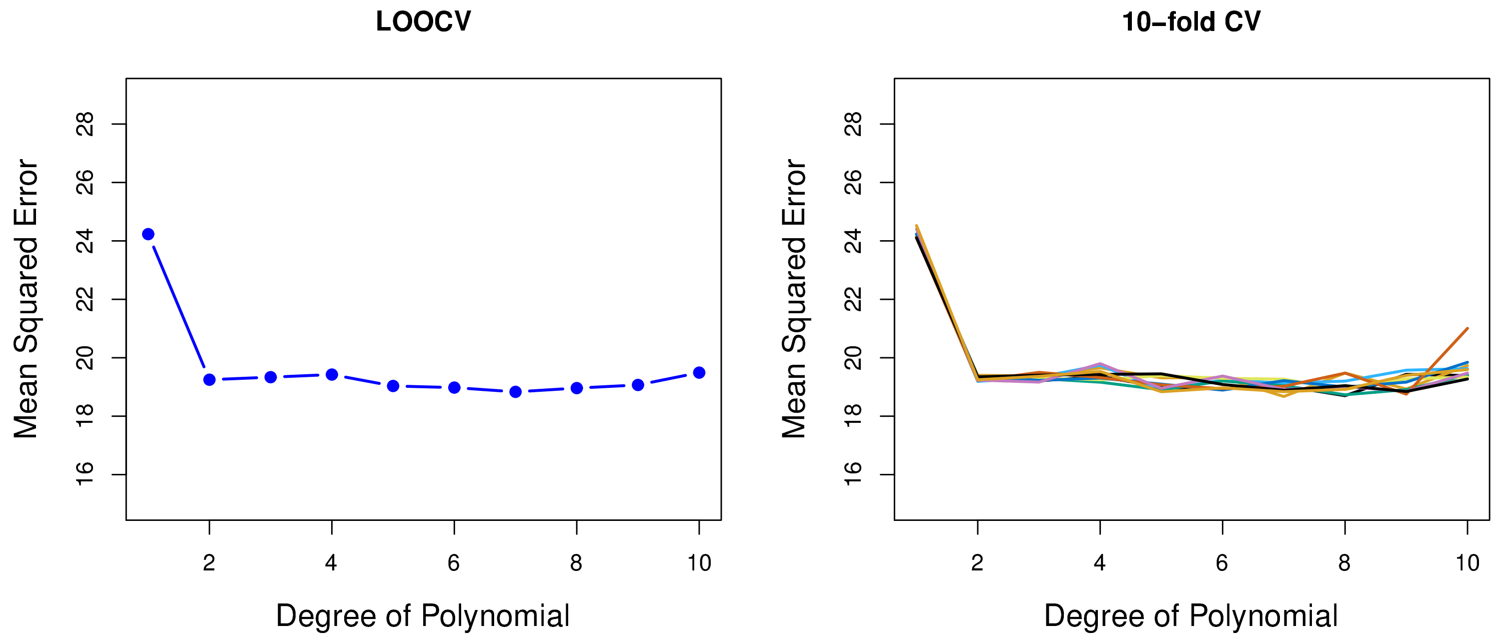
How to measure the complexity of a fitted model using the K nearest neighbours?

k-fold approach: randomly divide data into k groups (or folds) of equal size. Leave out one group each as a testing sample. Repeat this process k times.



A version of leave-$\frac{n}{k}$-out approach

The leave-one-out CV is unique while k-fold CV is not, as the way of dividing the whole data into k groups is not unique.



Cross validation is applied to `Auto.txt` with models

$$\texttt{mpg} = \beta_0 + \beta_1 \texttt{hpower} + \cdots + \beta_p \texttt{hpower}^p + \varepsilon$$

for $p = 1, \cdots, 10$.

Left panel: CV error curve: plot of CV-MSE vs p for leave-one-out approach

Right panel: 10-fold CV was run 9 times, each with a different random split of the data into 10 parts.

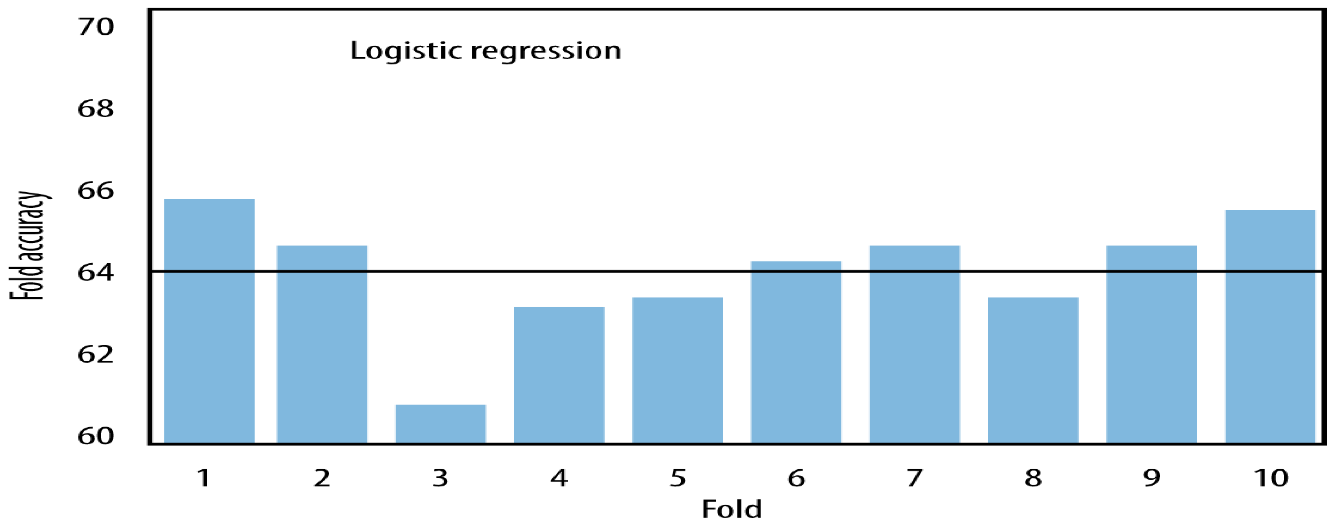
Churn example revisited

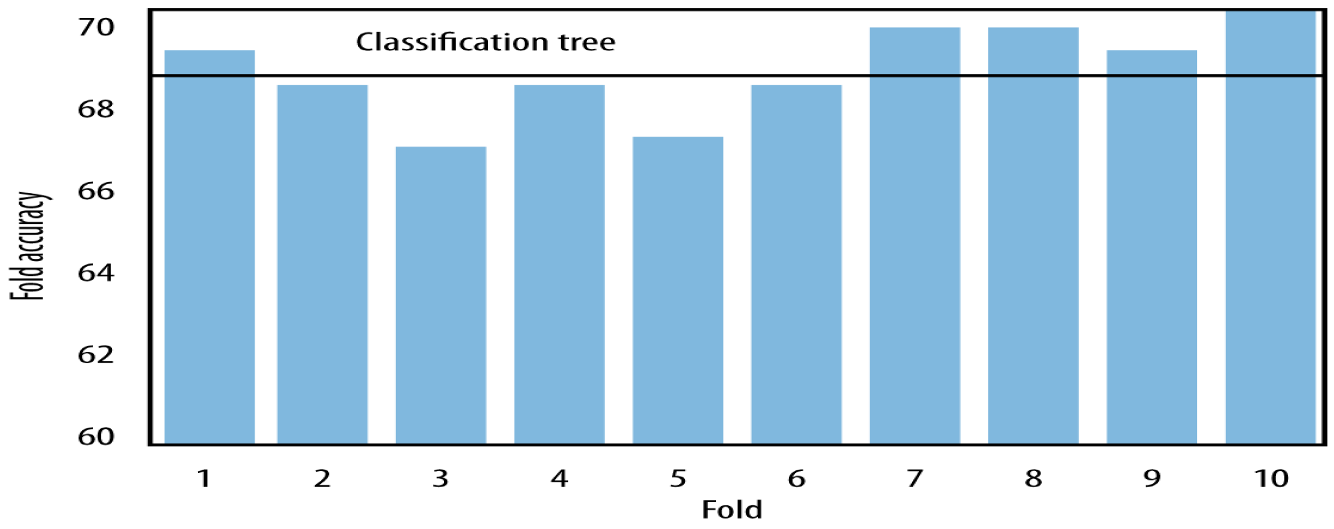
Section 4 derived a classification tree used the entire dataset for both training and testing with the reported 73% accuracy.

Now we run 10-fold cross-validation to select decision trees and logistic regression models: the whole data set is randomly divided into 10 folds. Each fold in turn served as a single holdout set while the other nine were collectively used for training.

The resulting classification: the majority votes of the 10 models.

The horizontal line is the average of accuracies of the 10 models in each panel.


Logistic regression
Fold accuracy
70
68
66
64
62
60
1
2
3
4
5
6
7
8
9
10
Fold


Classification tree
Fold accuracy
70
68
66
64
62
60
1
2
3
4
5
6
7
8
9
10
Fold

- The average accuracy of the 10 folds with classification trees is 68.6%; significantly lower than 73%, 68.6% is a more realistic indicator for the accuracy when the method applies in the real world

- For this particular dataset, tree method performs better than logistic regression (with the average accuracy 64.1%). Also the accuracy variation of the tree method over the 10 folds is smaller or much smaller: the standard deviations are 1.1% (for trees) and 1.3% (for logistic regression).

- The performance fluctuations of the two methods show the similar pattern: worst for Fold 3 and best for Fold 10.

Minimize expected MSE: An alternative approach to the (computationally intensive) validation methods

Consider linear regression:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$$

There are p candidate regressors $\mathbf{X} = (X_1, \cdots, X_p)$.

The model complexity is measured by the number of regressors used in a fitted model, say, d, $0 \le d \le p$

For given d, let $\widehat{f}_d$ denote the optimal subset regression with d regressors.

Ideally we should choose d to minimize the theoretical mean squared error:

$$\text{MSE}(d) = E[\{Y - \widehat{f}_d(\mathbf{X})\}^2]$$

Since it is unknown, we use an unbiased estimator

$$C_p(d) = \frac{1}{n}\{\mathsf{RSS}(d) + 2d\hat{\sigma}^2\},$$

where $\hat{\sigma}^2$ is the estimated variance for ε with the full model (i.e. p regressors).

C_p-**criterion**: choose the optimal subset regression with $\hat{d}$ regressors which minimizes $C_p(d)$.

Note. C_p-criterion is not applicable when p is large in relation to n!

Rule of thumb: $p \leq n/3$ or $n/4$.

Akaike information criterion (AIC)

Let $f_d(\mathbf{z}, \boldsymbol{\theta}_d)$ be a family of likelihood with the complexity indexed by d. Suppose there are available both training data and validation data, both with the sample size n.

- estimate $\boldsymbol{\theta}_d$ by maximising $\sum_{\mathbf{z}} \log f_d(\mathbf{z}, \boldsymbol{\theta}_d)$, leading to MLE $\widehat{\boldsymbol{\theta}}_d$, where the sum is taken over all $\mathbf{z}$ in the training data set
- choosing d to maximize $\sum_{\mathbf{z}} \log f_d(\mathbf{z}, \widehat{\boldsymbol{\theta}}_d)$, where the sum is taken over all $\mathbf{z}$ in the validation data set.

In practice we only have one data set, $-\frac{2}{n}\sum_{\mathbf{z}} \log f_d(\mathbf{z}, \widehat{\boldsymbol{\theta}}_d)$ has the same asymptotic mean as

$$\text{AIC} = -\frac{2}{n}(\text{maximized log likelihood}) + \frac{2}{n}(\text{No. of estimated parameters})$$

For regression model with normal errors,

$$\text{AIC}(d) = \log(\widehat{\sigma}_d^2) + 2d/n,$$

where $\widehat{\sigma}_d^2$ is the MLE for σ^2 in the optimal subset regression with d regressors.

AIC: choose d which minimizes AIC(d).

Bayesian information criterion (BIC)

$$\text{BIC} = -\frac{2}{n}(\text{maximized log likelihood}) + \frac{\log n}{n}(\text{No. of estimated parameters})$$

For regression model with normal errors,

$$\text{BIC}(d) = \log(\widehat{\sigma}_d^2) + (\log n)d/n,$$

where $\widehat{\sigma}_d^2$ is the MLE for σ^2 in the optimal subset regression with d regressors.

BIC: choose d which minimizes BIC(d).

Note. AIC tends to overestimate the number of regressors, while BIC gives a consistent estimator for the true d (i.e. the estimator converges to the true d when $n \to \infty$ but d is fixed).

Control complexity

A general principle: A good statistical model should provide an adequate fit to the available data, and should be as simple as possible.

Therefore an optimum model can be defined as the trade-off between *the goodness-of-fit* and *complexity of model*, i.e. it minimizes

$$\text{(Goodness of fit of the model)} + \text{(Penalty for model complexity)}$$

Two terms move to opposite directions when model complexity increases: GOF $\searrow$, Penalty $\nearrow$.

C_p-criterion, AIC and BIC are all of this form.

A criterion to pruning decision trees: for a given penalty constant $\lambda > 0$, search for the tree which minimizes

$$F_{\text{obj}} \equiv (\text{misclassification rate}) + \lambda \times (\text{number of nodes})$$

Remark. (i) λ controls the size of the tree. It can be selected by, for example, cross-validation or multi-fold cross validation as follows:

> Choose a grid of λ values, and compute the cross-validation version of F_{obj} for each value of λ. We then select the value of λ for which the cross-validation version of F_{obj} is smallest. Finally, the model is re-fit using all of the available observations and the selected value of λ.

(ii) An alternative approach is to require that each (terminal) node contains at least k data points, where $k \geq 1$ is an integer which controls the size of the tree.

Ridge Regression: an L_2 shrinkage method

For a regression model:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{pi} + \varepsilon_i, \quad i = 1, \cdots, p,$$

The ridge regression estimators for $\beta_0, \cdots, \beta_p$ are obtained by minimizing

$$\sum_{i=1}^{n} \Big(y_i - \sum_{j=1}^{p} \beta_j x_{ij} \Big)^2 + \lambda \sum_{j=1}^{p} \beta_j^2,$$

where $\lambda > 0$ is a tuning parameter controlling the degree of shrinkage: the larger λ is the smaller $|\beta_j|$ are.

The advantage of ridge regression is measured by MSE($=$ Var $+$ bias^2): increasing λ leads to decrease of variance and increase of bias. With appropriate values of λ, the MSE of ridge regression estimator can be smaller than that of the OLS.

Choosing λ: cross validation or multi-fold CV.

LASSO: L_1 shrinkage.

Ridge regression makes $|\beta_j|$ smaller while the resulting model typically contain all the p regressors.

LASSO changes the L_2 norm in the penalty to the L_1 norm:

$$\sum_{i=1}^{n}\Big(y_i - \sum_{j=1}^{p}\beta_j x_{ij}\Big)^2 + \lambda\sum_{j=1}^{p}|\beta_j|,$$

LASSO shrinks small β_j to 0, hence it also serves a variable/feature selection procedure.

LARS algorithm: compute LASSO solution path for all possible values of λ.

Final remark. Even proper validation is performed, a fitted model can still perform poorly in practice. They are many possible reasons:

- Data used in fitting the model do not match well the actual use scenario.

- Non-stationarity: the world has changed since the data used in fitting the model were collected.

- All candidate models are inadequate, running into the problem due to *multiple comparisons*

No silver bullet or magic recipe to truly get ‘the optimal’ model unfortunately! Also look into the 2nd or 3rd ‘best’ model.

We illustrate the R-implementation of some methods discussed in this chapter using the dataset `Hitter` in *ISLR* library: records on 20 variables from 322 major league baseball players in 1986/7 season.

```
> library(ISLR)
> names(Hitters)
 [1] "AtBat"     "Hits"      "HmRun"     "Runs"      "RBI"       "Walks"     "Years"
 [8] "CAtBat"    "CHits"     "CHmRun"    "CRuns"     "CRBI"      "CWalks"    "League"
[15] "Division"  "PutOuts"   "Assists"   "Errors"    "Salary"    "NewLeague"
> dim(Hitters)
[1] 322  20
> sum(is.na(Hitters))
[1] 59  # there are 59 missing values. Also try ‘> is.na(Hitters)’ directly
> Hitters1=na.omit(Hitters) # remove the rows with missing values
> dim(Hitters1)
[1] 263  20
```

Function `regsubsets` function (part of the `leaps` library) performs best subset selection by identifying the best model that contains a given number of predictors, where best is in the sense of minimum RSS.

```
> install.packages("leaps")
> library(leaps)
> subset.Hitter = regsubsets(Salary~., data=Hitters1)
> summary(subset.Hitter)
```

```
Subset selection object
Call: regsubsets.formula(Salary ~ ., data = Hitters1)
19 Variables  (and intercept)
1 subsets of each size up to 8
Selection Algorithm: exhaustive
         AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI CWalks League
1  ( 1 ) " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "
2  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "
3  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "
4  ( 1 ) " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "
5  ( 1 ) "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"  " "    " "
6  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"  " "    " "
7  ( 1 ) " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "  " "    " "
8  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "  "*"    " "
         DivisionW PutOuts Assists Errors NewLeagueN
1  ( 1 ) " "       " "     " "     " "    " "
2  ( 1 ) " "       " "     " "     " "    " "
3  ( 1 ) " "       "*"     " "     " "    " "
4  ( 1 ) "*"       "*"     " "     " "    " "
5  ( 1 ) "*"       "*"     " "     " "    " "
6  ( 1 ) "*"       "*"     " "     " "    " "
7  ( 1 ) "*"       "*"     " "     " "    " "
8  ( 1 ) "*"       "*"     " "     " "    " "
```

For example, the best model with 3 regressors selected `Hits`, `CRBI`, `PutOuts`.

```
> subset3.Hitters=lm(Salary~Hits+CRBI+PutOuts, Hitters1)
> summary(subset3.Hitters)

Call:
lm(formula = Salary ~ Hits + CRBI + PutOuts, data = Hitters1)

Residuals:
    Min      1Q  Median      3Q     Max 
-856.54 -171.37  -24.87  103.22 2169.87 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -71.45922   55.20273  -1.294 0.196650    
Hits          2.80382    0.49229   5.695 3.33e-08 ***
CRBI          0.68253    0.06584  10.366  < 2e-16 ***
PutOuts       0.27358    0.07778   3.517 0.000514 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 336.1 on 259 degrees of freedom
Multiple R-squared:  0.4514,Adjusted R-squared:  0.4451 
F-statistic: 71.05 on 3 and 259 DF,  p-value: < 2.2e-16
```

To see the info in the output

```
> names(summary(subset.Hitter))
[1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
> summary(subset.Hitter)$rsq # regression coefficient R^2
[1] 0.3214501 0.4252237 0.4514294 0.4754067 0.4908036 0.5087146 0.5141227 0.5285569
> summary(subset.Hitter)$bic  # BIC values for 8 selected models
[1]  -90.84637 -128.92622 -135.62693 -141.80892 -144.07143 -147.91690
[7]  -145.25594 -147.61525
> summary(subset.Hitter)$cp  # C_p values of 8 selected models
[1] 104.281319  50.723090  38.693127  27.856220  21.613011  14.023870
[7] 13.128474   7.400719
```

The squared regression correlation coefficient R^2 increases when the number of regressors increases.

BIC selects the best subset regression with 6 regressors

```
> coef(subset.Hitter, 6)
(Intercept)        AtBat         Hits        Walks         CRBI    DivisionW      PutOuts
 91.5117981   -1.8685892    7.6043976    3.6976468    0.6430169 -122.9515338    0.2643076
```

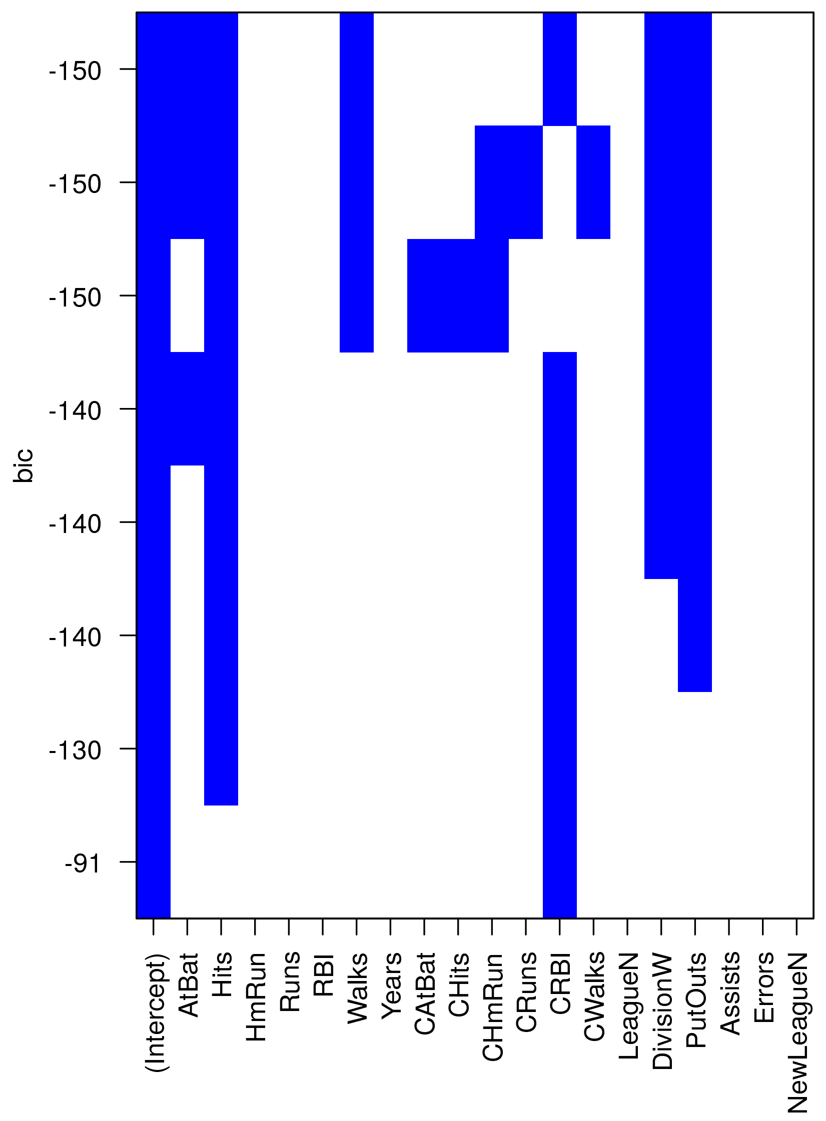
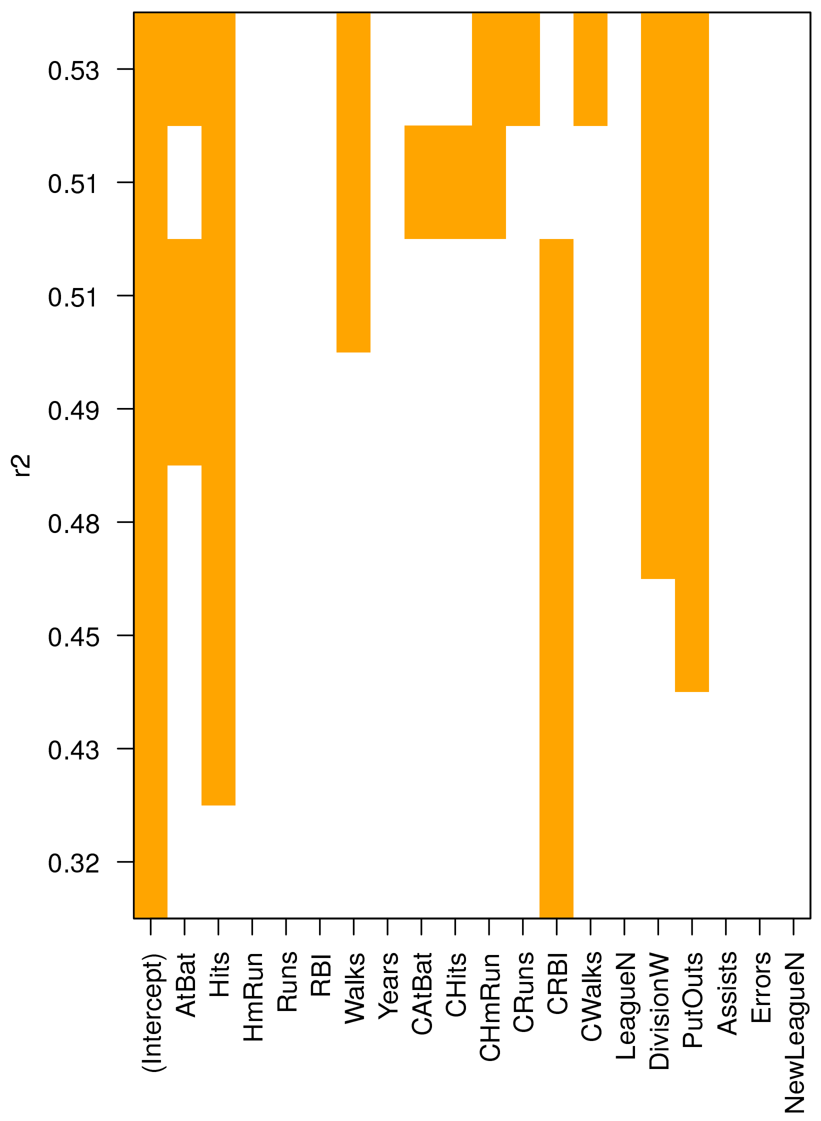
C_p selects the best subset regression with 8 regressors???

One can fit the models with the maximum 19 variables for this dataset:

```
> regsubsets(Salary ., data=Hitters1, nvmax=19)
```

`regsubsets` has a built-in plots which display the selected variables for the best model with a given number of predictors, ranked according to the BIC, Cp, R^2 and etc.

```
> par(mfrow=c(1,2))
> plot(subset.Hitter, scale="bic", col="blue")
> plot(subset.Hitter, scale="r2", col="orange")
```


bic
-150
-150
-150
-140
-140
-140
-130
-91
(Intercept)
AtBat
Hits
HmRun
Runs
RBI
Walks
Years
CAtBat
CHits
CHmRun
CRuns
CRBI
CWalks
LeagueN
DivisionW
PutOuts
Assists
Errors
NewLeagueN

r2
0.53
0.51
0.51
0.49
0.48
0.45
0.43
0.32
(Intercept)
AtBat
Hits
HmRun
Runs
RBI
Walks
Years
CAtBat
CHits
CHmRun
CRuns
CRBI
CWalks
LeagueN
DivisionW
PutOuts
Assists
Errors
NewLeagueN

Now we apply 10-fold CV to select variables. First we divide randomly the original 253 observations into 10 folds of (about) equal size

```
> folds=rep(1:10, 26) # repeat the sequence {1, ..., 10} 26 times
> length(folds)
> [1] 260
> folds=folds[1:253]  # cut it at length 253
> folds=sample(folds, 253, replace=F)  # change to random order
> folds
  [1] 6 1 7 4 8 9 2 6 2 2 6 2 7 3 1 4 10 2 7 1 8 6 2 7 9 5 5 5 5 5
 ... ...
 #  Observation 1 is in 6th fold, 2 in 1st fold, 3 in 7th fold ..
```

We fit the models using the data in 9 folds, and calculate the CV-RSS using the data in the fold which was left out.

Since `regsubsets` does not have a build-in prediction function, please download the file 'predict.regsubsets.r' from the course moodle page, and put it in your working directory.

```
> source("predict.regsubsets.r")
> cv.errors=matrix(nrow=10, ncol=19)
> for(j in 1:10) {
+      best.fit=regsubsets(Salary~., data=Hitters1[folds!=j,], nvmax=19)
+       for(i in 1:19) {
+            pred=predict(best.fit, Hitters1[folds==j,], id=i)
+            cv.errors[j,i]=mean((Hitters1$Salary[folds==j]-pred)^2)
+       }
+ }
```

To find the number of regressors which entails the minimum CV-errors across 10 folds

```
> cvErrors=apply(cv.errors, 2, mean) # apply mean to each column
> cvErrors
       1        2        3        4        5        6        7        8        9
151805.0 130218.6 140482.6 132581.5 132990.3 129238.9 124624.2 122341.4 124876.7
      10       11       12       13       14       15       16       17       18
123263.0 124850.5 124516.8 124608.5 125259.7 126883.1 127252.0 127290.5 126773.5
      19
126910.8

> n=1:19    # n is a vector with elements 1,...,19
> n[cvErrors[n] == min(cvErrors)]
> [1] 8     # the value n such that cvErrors[n]=min
```

Thus the 10-fold CV selects the best subset model with 8 predictors. The final selected model should be the one with 8 predictors/regressors but fitted using the whole data set.

Note. Repeating the above computation may lead to a different model (i.e. with a different number of predictors), as the division of the 10 folds is random.

If one set random seed fixed in the beginning of computation, say `set.seed(3)`, the same results will be repeated on the same computer.

To illustrate the application of CV for selecting tree models, we recall some of our treatment of dataset `Carseats` in Chapter 3:

```
> attach(Carseats)
> High=ifelse(Sales<=8, "No", "Yes")    # Define the label High iff Sales >8
> library(tree)
> Carseats2=data.frame(Carseats, High)   # combine the label into the data set
> tree.carseats=tree(High~.-Sales, Carseats2) # . indicates using all the predictors,
                                             # -Sales: exclude Sales
> summary(tree.carseats)
Classification tree:
tree(formula = High ~ . - Sales, data = Carseats2)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
[6] "Advertising" "Age" "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
```

Now we apply CV to determine the tree size.

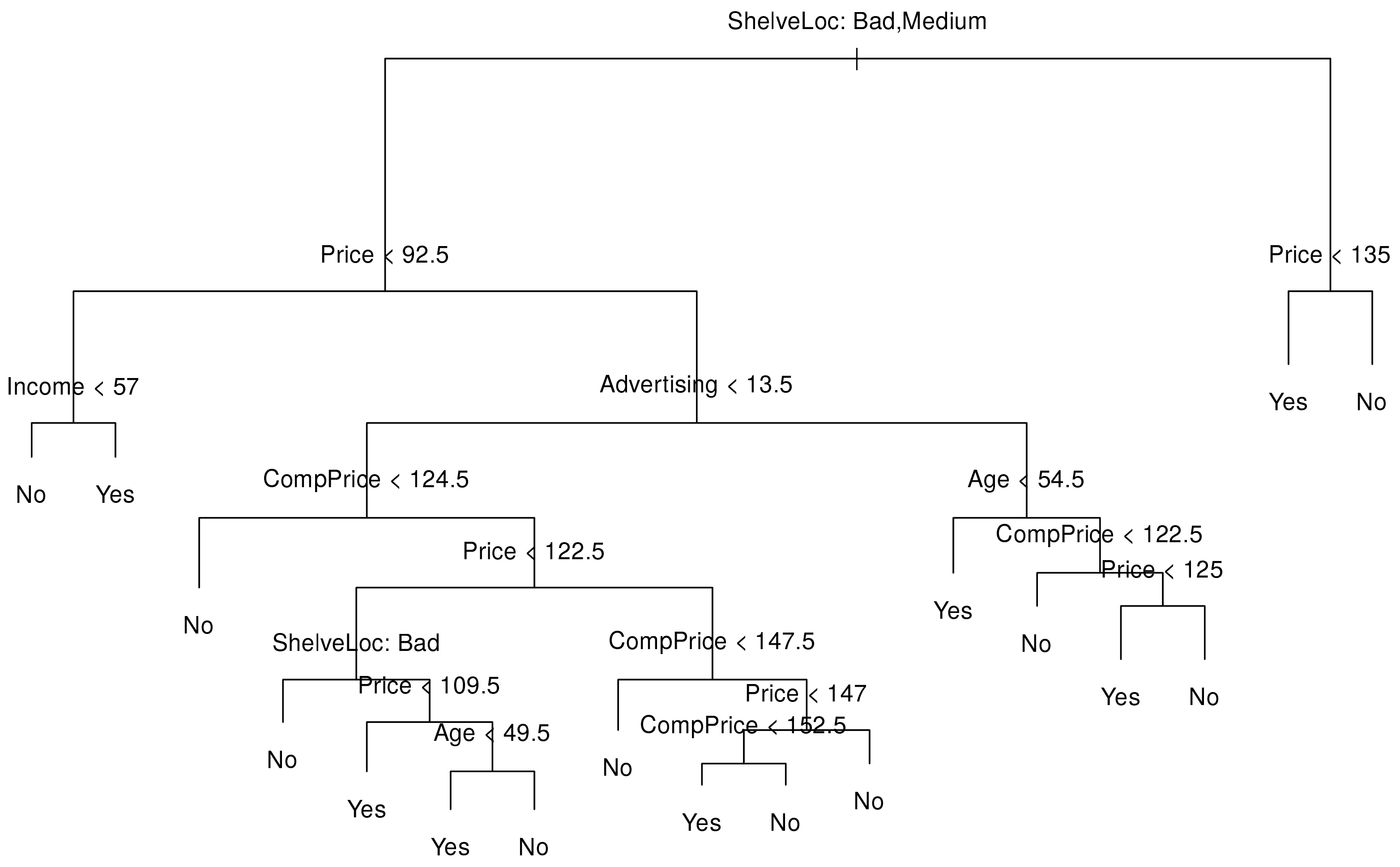
```
> cv.carseats=cv.tree(tree.carseats, FUN=prune.misclass)
> cv.carseats$size   # Number of terminal nodes
 [1] 27 26 24 22 19 17 14 12  7  6  5  3  2  1
> cv.carseats$dev    # Number of CV-misclassfications
```

```
 [1] 104 105 103 102 102 101 102 102 108 107 105 108 117 165
```

The minimum CV-value is 101, the corresponding tree has 17 terminal nodes. Now we apply function `prune.misclass` to prune the tree to obtain the nine node tree

```
> prune.carseats=prune.misclass(tree.carseats, best=17)
> plot(prune.carseats)
> par(mfrow=c(1,1))
> plot(prune.carseats)
> text(prune.carseats, pretty=0)
```

The resulting tree is much simpler and more interpretable than that in Chapter 3.


ShelveLoc: Bad,Medium
Price < 92.5
Price < 135
Yes
No
Income < 57
No
Yes
Advertising < 13.5
CompPrice < 124.5
No
Price < 122.5
ShelveLoc: Bad
No
Price < 109.5
Yes
Age < 49.5
Yes
No
CompPrice < 147.5
No
Price < 147
CompPrice < 152.5
Yes
No
No
Age < 54.5
Yes
CompPrice < 122.5
No
Price < 125
Yes
No