Chapter 2
Model selection

Simon Fraser University
ECON 483
Summer 2023

Disclaimer

These notes are based on the Book **Introduction to Statistical Learning with R**, by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. Any error is my sole responsibility.

What is model selection?

- Predictions can be built in many different ways: Different methods, different variables, etc
- The use of some estimators over others can be justified by the assumptions made: OLS, IV, GLS, etc
- But once one settles on an estimator, what variables should one include, besides the ones of interest?
- In this lecture, we are going to compare models in order to select the “best” one
- Model selection is more important for **prediction** problems than **estimation** problems as in estimation problems, we are interested in the effect of a specific variable
- Suggested reading: Chapter 6 in **ISLR**

Measuring the quality of Fit

- The sample used to estimate the model is called the **training sample**
- Recall the definition of in-sample MSE:

$$MSE \equiv \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

- The OLS estimator pretty much minimizes the **training sample MSE** assuming $f(x_i) = x_i'\beta$
- But at the end of the day, we don't want to predict the data we observe. Rather, we want to observe data where we only observe x_i, not y_i
- So minimizing the MSE of the sample we use to estimate the model does not tell much about the **general** quality of the fit

Outline

- The bias-variance tradeoff
    - Bias
    - Variance
- Selection of linear models
- Validation sets and cross validation
- Model averaging

The bias-variance tradeoff

The bias-variance tradeoff: Bias

- The **bias** of a statistical method refers to the error coming from approximating the real relationship by a simpler one
- If the real relationship is linear, a linear model will do good
- If not, a linear model will fail to pick up the true relationship pattern
- As flexibility increases, the **bias** of the method decreases

The bias-variance tradeoff: Variance

- One can show that we can always increase the fit of the **training data** by increasing flexibility
- But the predictions $\hat{f}()$ will vary a lot if I use another training data set: That statistical method has high **variance**
- Flexibility helps capture patterns on the training data set, but these patterns could be coming from that data set in particular, and not be present in all data sets following the same data generating process
- This is **overfitting!** (we can always increase the R^2 by increasing flexibility, i.e. the number of regressors)
- On the other hand, a rigid model fails to pick up the relevant patterns
- Not flexible models don't try hard enough (low variance), too flexible models try too hard (high variance)

The bias-variance tradeoff: MSE Decomposition

- Recall the population regression equation:

$$\boldsymbol{y_i = f(x_i) + u_i}$$

- One can decompose the **population test MSE** as:

$$\mathbb{E}[(\boldsymbol{y_0} - \hat{\boldsymbol{f}}(\boldsymbol{x_0}))^2] = \boldsymbol{Var}[\hat{\boldsymbol{f}}(\boldsymbol{x_0})] + (\mathbb{E}[\boldsymbol{f}(\boldsymbol{x_0}) - \hat{\boldsymbol{f}}(\boldsymbol{x_0})])^2 + \boldsymbol{Var}(\boldsymbol{u_0})$$

where $\boldsymbol{x_0}$ is an observation taken out of the training sample

- The first term is the **variance** of the estimator $\hat{\boldsymbol{f}}(\boldsymbol{x_0})$, the second term its squared **bias**, the third term is the **irreducible error**

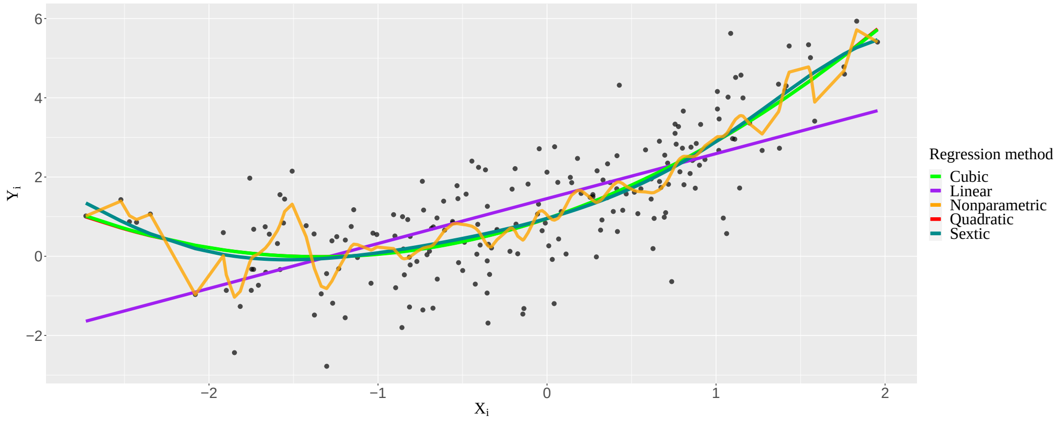
The bias-variance tradeoff: MSE Decomposition

$$\mathbb{E}[(\boldsymbol{y_0} - \hat{\boldsymbol{f}}(\boldsymbol{x_0}))^2] = \boldsymbol{Var}[\hat{\boldsymbol{f}}(\boldsymbol{x_0})] + (\mathbb{E}[\boldsymbol{f}(\boldsymbol{x_0}) - \hat{\boldsymbol{f}}(\boldsymbol{x_0})])^2 + \boldsymbol{Var}(\boldsymbol{u_0})$$

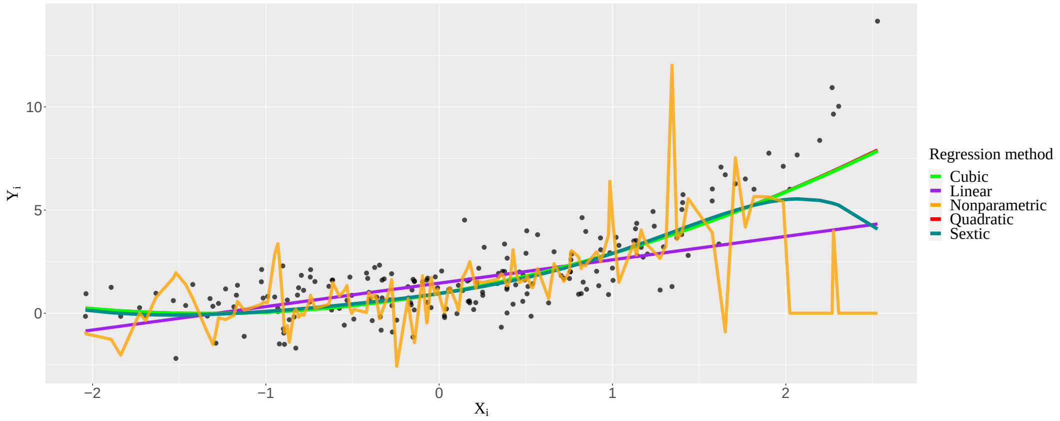
where $\boldsymbol{x_0}$ is an observation taken out of the training sample

- The first term is the **variance**, the second term the **bias** (squared), the third term is the **irreducible error** (cannot be reduced as it is the part of the equation the model does not capture)
- As flexibility of a method increases:
  - its **bias** decreases
  - its **variance** increases
- All the terms are positive, so we want all of them to be low: There is a **tradeoff!**

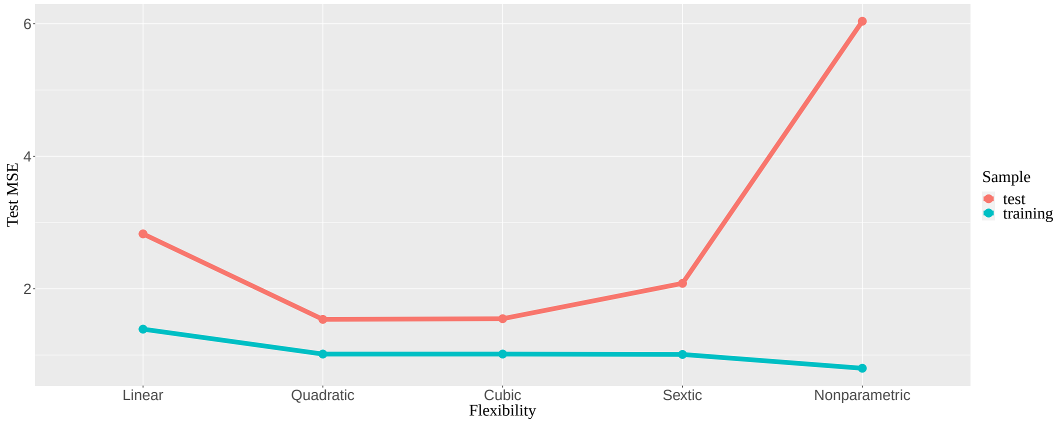
Training data predictions



Test data predictions



The bias-variance tradeoff: Illustration



The bias-variance tradeoff: Summary

- The **training data MSE** is a decreasing function of flexibility of the model
- In words: The more flexible the model, the better the predictions of the training sample
- The **test data MSE** has a U shape!
- In words: A very flexible model and a non flexible model do not predict as well as a moderately flexible model
- The different model selection methods presented here aim at finding the best **tradeoff**

Selection among linear models

Selection among linear models: Outline

- Information criteria
- Subset selection
  - Best subset selection
  - Stepwise selection
- Dimension reduction
  - Principal component analysis
  - Partial least squares

Information criteria

Information criteria

- For **linear models** with different numbers of variables, we can use criteria that account for the **bias-variance tradeoff** using the **full sample**, i.e. without dividing the sample into training vs test samples
- Let
$$\boldsymbol{RSS \equiv \sum_{i=1}^{n}(y_i - x_i'\hat{\beta})^2}$$
be the **Residual Sum of Squares** of the sample
- Let $\boldsymbol{\hat{\sigma}^2}$ be a consistent estimate of $\boldsymbol{\sigma^2}$, the variance of the error term
- Given that RSS decreases as the number of variables increases, one should include a "penalty term" for increasing flexibility to mitigate the variance of the model
- Criteria involving RSS and a penalty should be **minimized**, as we want both parts to be small

C_p, AIC, BIC, and adjusted R^2

- For a model employing d variables The C_p estimate is defined as

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

- The **Akaike information criterion (AIC)** is defined as

$$AIC(d) \equiv \frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma}^2)$$

- Note that AIC is used if the error term is assumed to follow a normal distribution, and is equivalent to C_p for linear models

C_p, AIC, BIC, and adjusted R^2

- The **Bayesian information criterion (BIC)** is defined as

$$BIC(d) \equiv \frac{1}{n\hat{\sigma}^2}(RSS + d\log(n)\hat{\sigma}^2)$$

- The **adjusted R^2** denoted $\bar{R}^2$ is defined as

$$\bar{R}^2 \equiv 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)}$$

- The **adjusted R^2** should be maximized, as it depends negatively on RSS

Subset selection

Best subset selection

- When one has many covariates available, the question is not only to know **how many** variables to include but also **which** to include
- If one has access to 5 covariates, it makes 1 model with 0 covariates, 5 models with 1 covariate, 10 models with 2 covariates, 10 models with 3 covariates, 5 models with 4 covariates, and 1 model with 5 covariates. Proceed as follows:
    - Find the best model **for each** number of covariates: the best model with 1 covariate, the best with 2, etc based on the best $\boldsymbol{R^2}$ every time
    - Find the best model overall based on $\boldsymbol{AIC}$, $\boldsymbol{BIC}$, $\boldsymbol{C_p}$, or adjusted $\boldsymbol{R^2}$

Algorithm: Best subset selection

- Find the best model **for each** number of covariates: the best model with 1 covariate ($\boldsymbol{\mathcal{M}_1}$), the best with 2 ($\boldsymbol{\mathcal{M}_2}$), etc based on the best $\boldsymbol{R^2}$ every time
- Find the best model overall based on $\boldsymbol{AIC}$, $\boldsymbol{BIC}$, $\boldsymbol{C_p}$, or adjusted $\boldsymbol{R^2}$

Stepwise selection: Forward and backward

- If one has many covariates, **best subset selection** becomes computationally inefficient
- Instead, one can gradually increase the number of covariates to add: This is **Stepwise forward selection**
  - Start with the smallest model (no covariates)
  - Choose the next variable to add based on highest increase in $\boldsymbol{R^2}$
- Choose the best model overall based on $\boldsymbol{AIC}$, $\boldsymbol{BIC}$, $\boldsymbol{C_p}$, or adjusted $\boldsymbol{R^2}$
- **Stepwise backward selection** works the opposite way: Start with the full model, remove covariates one by one based on highest $\boldsymbol{R^2}$, and finally choose the best model overall based on $\boldsymbol{AIC}$, $\boldsymbol{BIC}$, $\boldsymbol{C_p}$, or adjusted $\boldsymbol{R^2}$
- Both methods involve estimating less models than best subset selection

Stepwise selection: Forward

Algorithm: Forward selection

- Estimate the model without any covariate, call it $\boldsymbol{\mathcal{M}_0}$
- Find the best model that augments $\boldsymbol{\mathcal{M}_0}$ with one more covariate in terms of $\boldsymbol{R^2}$: call it $\boldsymbol{\mathcal{M}_1}$
- Find the best model that augments $\boldsymbol{\mathcal{M}_1}$ with one more covariate: call it $\boldsymbol{\mathcal{M}_2}$
- Find the best model that augments $\boldsymbol{\mathcal{M}_2}$ with one more covariate: call it $\boldsymbol{\mathcal{M}_3}$
- Continue until obtaining $\boldsymbol{\mathcal{M}_K}$
- Select the best model among $\boldsymbol{\mathcal{M}_0}$, $\boldsymbol{\mathcal{M}_1}$, $\boldsymbol{\mathcal{M}_2}$,...,$\boldsymbol{\mathcal{M}_K}$ using $\boldsymbol{AIC}$, $\boldsymbol{BIC}$, $\boldsymbol{C_p}$, or adjusted $\boldsymbol{R^2}$

Stepwise selection: Backward

Algorithm: Backward selection

- Estimate the full model, i.e. the one with all the K covariates, call it $\mathcal{M}_K$
- Find the best model that removes one covariate from $\mathcal{M}_K$ in terms of R^2: call it $\mathcal{M}_{K-1}$
- Find the best model that removes one covariate from $\mathcal{M}_{K-1}$: call it $\mathcal{M}_{K-2}$
- Continue until obtaining $\mathcal{M}_0$
- Select the best model among $\mathcal{M}_0$, $\mathcal{M}_1$, $\mathcal{M}_2$,...,$\mathcal{M}_K$ using AIC, BIC, C_p, or adjusted R^2

Subset selection in Rstudio

- Consider the **Credit** data set from the **ISLR** library
- Reports the **Balance** (average credit card debt. Not lower than zero here) for a number of individuals with:
  - Quantitative predictors: **age, education, income**
  - Qualitative predictors: **gender, student, status, ethnicity**
- We want to predict **Balance**: what combination of variables to include?

```
data(Credit)
Credit
n_obs <- nrow(Credit)
Credit_1 <- Credit[1:floor(n_obs/3),]
Credit_2 <- Credit[(floor(n_obs/3)+1):floor(2*n_obs/3),]
Credit_3 <- Credit[(floor(2*n_obs/3)+1):n_obs,]
Credit_12 <- rbind(Credit_1, Credit_2) # Combine the first 2 subsets
K <- ncol(Credit_12)  # number of variables
```

Best subset selection in R

- The **`regsubsets`** function from the **`leaps`** package allows to perform subset selection
- Several arguments must be specified
  - **`formula`**: Full model formula
  - **`data`**: Data frame (optional)
  - **`nvmax`**: Maximum size of subsets to consider (optional)
  - **`method`**: Forward, backward or exhaustive search can be done (optional)
- Many other options can be used, but they are not necessary for the function to run

Best subset selection in R

```r
subset_selection <- regsubsets(Balance ~ ., data = Credit_12, nvmax = 12)
# Show the different measures after all the models are estimated
subset_sum <- summary(subset_selection)
data.frame( Adj.R2 = which.max(subset_sum$adjr2),
            CP = which.min(subset_sum$cp),
            BIC = which.min(subset_sum$bic)     )
```

```
##   Adj.R2 CP BIC
## 1      6  5   4
```

```r
# Let us do best subset selection according to adjusted R squared
subset_adjr2_model <- data.frame(selected =
              as.matrix(subset_sum$which[which.max(subset_sum$adjr2), ]))
# Look at the selected variables
subset_adjr2_model <- dplyr::filter(subset_adjr2_model, selected == TRUE)
subset_adjr2_model$variable <- row.names(subset_adjr2_model)
subset_adjr2_model$variable
```

```
## [1] "(Intercept)" "Income"      "Limit"       "Rating"      "Cards"
## [6] "Age"         "StudentYes"
```

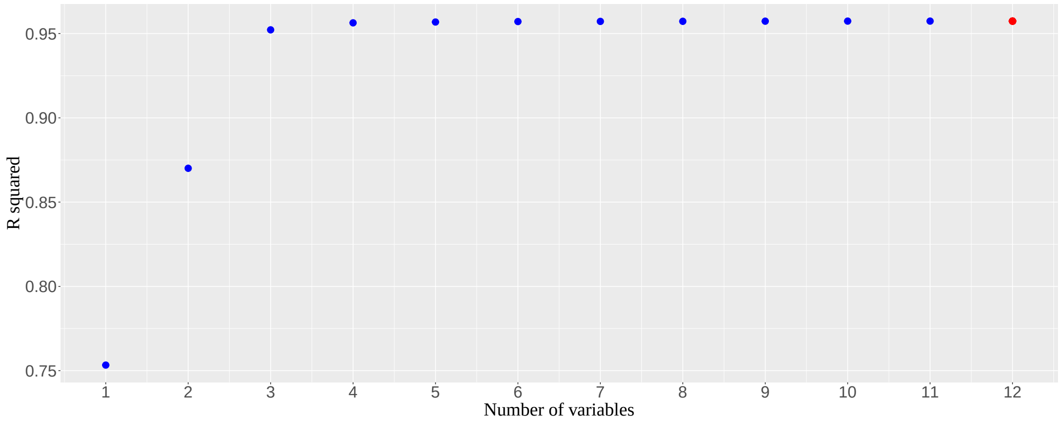
Best subset selection in R

```r
# Best subset selection model according to Adjusted R squared
best_subset_model <- lm(Balance ~ Income + Limit + Rating + Cards + Age + Student, data = Credit_3)
summary(best_subset_model)
```

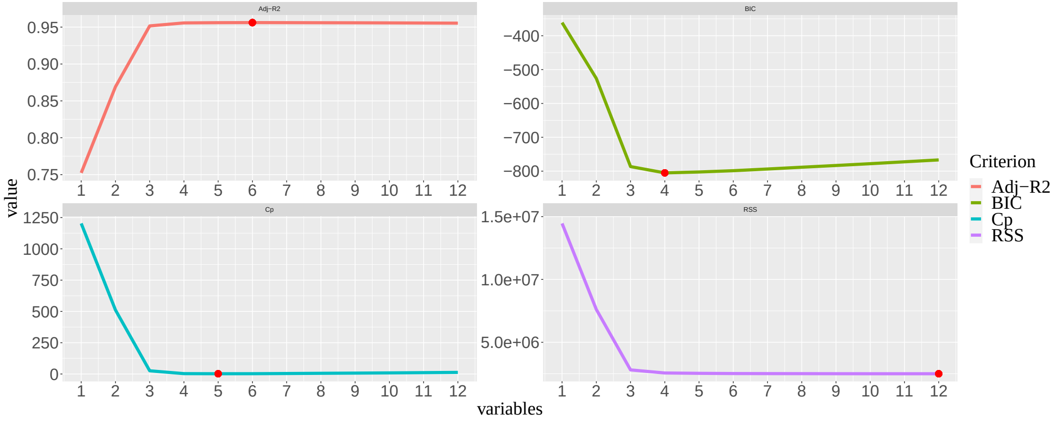
```
##
## Call:
## lm(formula = Balance ~ Income + Limit + Rating + Cards + Age +
##     Student, data = Credit_3)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -190.081  -65.887   -7.253   55.438  234.015
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -472.20437   43.78068 -10.786  < 2e-16 ***
## Income        -7.42758    0.39797 -18.663  < 2e-16 ***
## Limit          0.14978    0.05686   2.634  0.00949 **
## Rating         1.59622    0.85905   1.858  0.06547 .
## Cards         17.52667    7.39134   2.371  0.01923 *
## Age           -0.57408    0.51206  -1.121  0.26435
## StudentYes   397.52867   31.43698  12.645  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 98.61 on 127 degrees of freedom
## Multiple R-squared:  0.9519, Adjusted R-squared:  0.9496
## F-statistic: 418.9 on 6 and 127 DF,  p-value: < 2.2e-16
```

```r
# Predict Balance in the third data set
best_subset_pred <- predict(best_subset_model, newdata = Credit_3)
```

Best subset selection in R



Best subset selection in R



Forward stepwise selection in R

```r
forward_selection <- regsubsets(Balance ~ ., data = Credit_12, nvmax = 12,
                                method = "forward")
forward_sum <- summary(forward_selection)
data.frame(
  Adj.R2 = which.max(forward_sum$adjr2),
  CP = which.min(forward_sum$cp),
  BIC = which.min(forward_sum$bic)
)
```

```
##   Adj.R2 CP BIC
## 1      6  6   5
```

```r
# Let us do forward selection according to BIC
forward_bic_model <- data.frame(selected =
                     as.matrix(forward_sum$which[which.min(forward_sum$bic),  ]))
# Look at the selected variables
forward_bic_model <- dplyr::filter(forward_bic_model, selected == TRUE)
forward_bic_model$variable <- row.names(forward_bic_model)
forward_bic_model$variable
```

```
## [1] "(Intercept)" "Income"      "Limit"       "Rating"      "Cards"
## [6] "StudentYes"
```

Forward stepwise selection in R

```r
# Best forward stepwise selection model according to BIC
forward_stepwise_model <- lm(Balance ~ Income + Limit + Rating + Cards + Student, data = Credit_3)
summary(forward_stepwise_model)
```

```
##
## Call:
## lm(formula = Balance ~ Income + Limit + Rating + Cards + Student,
##     data = Credit_3)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -201.240  -70.724   -3.446   63.156  246.678
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -501.9577    34.8550 -14.401   <2e-16 ***
## Income        -7.5171     0.3903 -19.262   <2e-16 ***
## Limit          0.1456     0.0568   2.564   0.0115 *
## Rating         1.6640     0.8578   1.940   0.0546 .
## Cards         16.4299     7.3337   2.240   0.0268 *
## StudentYes   398.5609    31.4550  12.671   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 98.71 on 128 degrees of freedom
## Multiple R-squared:  0.9514, Adjusted R-squared:  0.9495
## F-statistic: 501.4 on 5 and 128 DF,  p-value: < 2.2e-16
```

```r
# Predict the third data set
forward_stepwise_pred <- predict(forward_stepwise_model, newdata = Credit_3)
```

Backward stepwise selection in R

```r
backward_selection <- regsubsets(Balance ~ ., data = Credit_12, nvmax = 12,
                                 method = "backward")
backward_sum <- summary(backward_selection)
data.frame(Adj.R2 = which.max(backward_sum$adjr2),
           CP = which.min(backward_sum$cp),
           BIC = which.min(backward_sum$bic) )
```

```
##   Adj.R2 CP BIC
## 1      6  5   4
```

```r
# Let us do backward selection according to Cp
backward_cp_model <- data.frame(selected =
                     as.matrix(backward_sum$which[which.min(forward_sum$cp),  ]))
# Look at the selected variables
backward_cp_model <- dplyr::filter(backward_cp_model, selected == TRUE)
backward_cp_model$variable <- row.names(backward_cp_model)
backward_cp_model$variable
```

```
## [1] "(Intercept)" "Income"      "Limit"       "Rating"      "Cards"
## [6] "Age"         "StudentYes"
```

Backward stepwise selection in R

```r
# Best backward stepwise selection model according to Cp
backward_stepwise_model <- lm(Balance ~ Income + Limit + Cards + Age, data = Credit_3)
summary(backward_stepwise_model)
```

```
##
## Call:
## lm(formula = Balance ~ Income + Limit + Cards + Age, data = Credit_3)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -234.21 -102.88  -28.68   68.58  510.67
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.498e+02  5.472e+01  -6.392 2.73e-09 ***
## Income      -7.365e+00  5.988e-01 -12.299  < 2e-16 ***
## Limit        2.516e-01  9.267e-03  27.147  < 2e-16 ***
## Cards        2.100e+01  9.205e+00   2.282   0.0241 *
## Age         -8.892e-01  7.755e-01  -1.147   0.2537
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 149.8 on 129 degrees of freedom
## Multiple R-squared:  0.8872, Adjusted R-squared:  0.8837
## F-statistic: 253.7 on 4 and 129 DF,  p-value: < 2.2e-16
```

```r
# Predict the third data set
backward_stepwise_pred <- predict(backward_stepwise_model, newdata = Credit_3)
```

Dimension reduction

Dimension reduction

- The previous approaches select covariates (information criteria, subset selection)
- Another approach is to "represent" many variables by transforming them into a set of variables with similar features, but in a lesser number
- The model is then estimated using the transformed variables
- The transformed variables have the most important features of the original variables, but they will help mitigate **overfitting** by reducing the dimension of the original set
- It is particularly relevant in the presence of collinearities, or if the number of covariates is close/bigger than the sample size
- Note that if the variables are transformed, they lose interpretation. So transformations can be used for prediction, not estimation

Dimension reduction: Principal components regression

- **Principal Component Analysis (PCA)** is an **unsupervised learning** method (no $\boldsymbol{Y}$ involved, only $\boldsymbol{X}$'s)
- Generally used to observe high-dimensional data before getting into further analysis
- It summarizes variance and correlation patterns of covariates in a more compact way
- With $\boldsymbol{p}$ variables at hand, **Principal Component Analysis (PCA)** looks for $\boldsymbol{M \ll p}$ variables $\boldsymbol{Z_m}$ that capture the main features (in particular, variance and correlation) of the original variables:

$$\boldsymbol{Z_m = \sum_{j=1}^{p} \phi_{j,m} X_j}$$

Dimension reduction: Principal components regression

$$\boldsymbol{Z_m = \sum_{j=1}^{p} \phi_{j,m} X_j}$$

- PCA finds the coefficients $\boldsymbol{\phi_{j,m},\ j = 1, ..., p,\ \ m = 1, ..., M}$
- **Principal components regression** then consists in estimating the model using these $\boldsymbol{M}$ variables
- How much should $\boldsymbol{M}$ be is found via cross validation (again!)

PCA in Rstudio

- In **Rstudio**, the **`pls`** package proposes principal component regression. The command to use is **`pcr()`**
- We can use cross validation to select the optimal amount of principal components to use (hopefully less than $\boldsymbol{p}$, so dimension reduction actually happens)

```r
# Principal component regression
pcr_fit <- pcr(Credit_12$Balance ~ data.matrix(Credit_12[ , 2:(K - 1)]), scale = TRUE,
               validation = "CV")
```

PCA in Rstudio

```
# Principal component regression
pcr_fit <- pcr(Credit_12$Balance ~ data.matrix(Credit_12[ , 2:(K - 1)]), scale = TRUE,
               validation = "CV")
summary(pcr_fit)
```

```
## Data:    X dimension: 266 10
##  Y dimension: 266 1
## Fit method: svdpc
## Number of components considered: 10
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           471.1    296.2    296.6    294.3    292.7    283.3    281.5
## adjCV        471.1    295.8    298.0    292.8    293.7    283.5    280.4
##        7 comps  8 comps  9 comps  10 comps
## CV       276.2    276.5    101.4     101.3
## adjCV    275.6    275.9    101.2     101.1
##
## TRAINING: % variance explained
##                   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## X                   27.44    39.13    50.26    60.95    71.35    80.73
## Credit_12$Balance   60.70    60.74    63.69    63.78    66.55    67.56
##                   7 comps  8 comps  9 comps  10 comps
## X                   89.24    97.17    99.98    100.00
## Credit_12$Balance   68.50    68.86    95.65     95.73
```

```
# validationplot(pcr_fit, scale = TRUE, val.type = "MSEP")
pcr_pred <- predict(pcr_fit, data.matrix(Credit_3[ , 2:(K - 1)]), ncomp = 7)
```

Dimension reduction: Partial least squares

- PCA is **unsupervised**, partial least squares is **supervised**: It uses the dependent variable $\boldsymbol{Y}$
- Idea: look for $\boldsymbol{M \ll p}$ variables $\boldsymbol{Z_m}$ that capture the main features (in particular, variance and correlation) of the original variables:

$$\boldsymbol{Z_m = \sum_{j=1}^{p} \phi_{j,m} X_j}$$

- PLS uses $\boldsymbol{Y}$ in the process of finding $\boldsymbol{\phi_{j,m},\ j = 1, \ldots, p,\ \ m = 1, \ldots, M}$
- This way, the new variables $\boldsymbol{Z_m}$ not only represent the old variables, but are also related to $\boldsymbol{Y}$

Partial Least Squares in **Rstudio**

- In **Rstudio**, the `pls` package proposes principal component regression. The command to use is `plsr()`
- We can use cross validation to select the optimal amount of principal components to use (hopefully less than p, so dimension reduction actually happens)

```r
# Partial least squares regression
plsr_fit <- plsr(Credit_12$Balance ~ data.matrix(Credit_12[ , 2:(K - 1)]), scale = TRUE,
                 validation = "CV")
# validationplot(plsr_fit, scale = TRUE, val.type = "MSEP")
plsr_pred <- predict(plsr_fit, data.matrix(Credit_3[ , 2:(K - 1)]), ncomp = 7)
```

Partial Least Squares in Rstudio

```r
# Partial least squares regression
summary(plsr_fit)
```

```
## Data:    X dimension: 266 10
##  Y dimension: 266 1
## Fit method: kernelpls
## Number of components considered: 10
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           471.1    259.0    170.2    102.4    100.9    100.9    100.9
## adjCV        471.1    258.6    169.3    102.0    100.8    100.7    100.8
##        7 comps  8 comps  9 comps  10 comps
## CV       101.0    100.3    100.3     100.3
## adjCV    100.7    100.2    100.2     100.2
##
## TRAINING: % variance explained
##                   1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## X                   27.01    35.20    40.95    50.13    59.80    68.89
## Credit_12$Balance   71.14    88.11    95.50    95.66    95.66    95.66
##                   7 comps  8 comps  9 comps  10 comps
## X                   71.84    79.78    89.32    100.00
## Credit_12$Balance   95.72    95.73    95.73     95.73
```

```r
# validationplot(plsr_fit, scale = TRUE, val.type = "MSEP")
# plsr_pred <- predict(plsr_fit, data.matrix(Credit_3[ , 1:(K - 1)]), ncomp = 7)
```

Validation sets and cross validation

Validation sets

- The methods above apply to linear models as the flexibility is dictated by the number of variables included
- Validation sets and cross validation work on a broader range of statistical methods, as they are just based on prediction. Hence they are widely used in the machine learning community
- Since we want to select a model based on out-of-sample performance, let us look at out-of-sample performance!
- Cut the data set into 2 data sets:
  - The **training data set** is used to estimate the models (estimate the OLS coefficients, etc)
  - The **test data set** is used to **test** the models predictions
- The best model is chosen based on the lowest **test MSE**

Validation sets: Illustration in Rstudio

The process goes as follows:

- Divide the data set in two data sets: **training** and **test** data sets (we divide it in 3 to see how the best model predicts the 3^{rd} data set)
- **"Train"** or **estimate** all the possible linear models on the **training data set**
- Compute the **test MSE** of each model using the **test data set**
- Choose the model with the lowest **test MSE**

Validation sets: Illustration in Rstudio

- In order to do this the least tedious way, I create several functions that will automatize the process
- First, a function that creates a formula when inputting the $\boldsymbol{X}$ and $\boldsymbol{Y}$ variables

```r
gen_formula <- function(y_name, X_names){
  formu <- as.formula(
    paste(y_name,"~",
          paste(X_names, collapse = " + ")) )
  return(formu)
}
```

```r
# Example
y_var <- "Income"
x_vars <- c("Age", "Experience", "Education")
gen_formula(y_var, x_vars)
```

```
## Income ~ Age + Experience + Education
## <environment: 0x0000000024a19b30>
```

Validation sets: Illustration in Rstudio

- Then, a function that generates a formula using the previous function, run a linear regression on ***training_data***, and computes the **training MSE** (***training_data***) and **test MSE** (***test_data***)

```r
# We make a function that computes the training and test MSE
# when y_name is the name of the dependent variable,
# and X_name are the names of the regressors
MSEs <- function(X_name, Y_name,  training_data, test_data)
  {
  form <- gen_formula( y_name = Y_name , X_names = X_name ) # Make the formula
  reg_results <- lm(form, data = training_data) # Regress the formula on the training data set

  df_training <- training_data %>%
    add_residuals(reg_results) %>%  # Adds a column of residuals to training_data called "resid"
    summarize( MSE = mean(resid^2) ) # Computes the MSE of the training sample
  training_MSE <- df_training[1,1]  # Get the training sample MSE as a number

  df_test <- test_data %>%
    add_residuals(reg_results) %>%  # Adds a column of residuals to test_data called "resid"
    summarize( MSE = mean(resid^2) ) # Computes the MSE of the test sample
  test_MSE <- df_test[1,1]  # Get the training sample MSE as a number
  k <- length(X_name)  # Report the number of X s
  return(c( k , training_MSE , test_MSE ))
}
```

Validation sets: Illustration in Rstudio

- Now, generate all the possible combinations of models with a provided set of variables. The function ***name_from_bin*** gathers the variable names corresponding to a sequence of 1 and 0 matching the ***vars*** vector of variable names. If it is a 1, show the variable, otherwise don't

```r
# Function to help get all the possible combination of variables
# It takes names from a vector according to where the 1 are
name_from_bin <- function(b, vars){
  return(vars[as.logical(b)])
}
```

```r
# Example
X <- c("age", "ethnicity", "marital status")
selector <- c(1, 0, 1) # We want to get the first and third characters
name_from_bin(selector, X)
```

```
## [1] "age"            "marital status"
```

Validation sets: Illustration in Rstudio

- And ***all_models*** makes all the possible combinations of $\boldsymbol{X}$'s we provide, and returns a list of combinations

```r
# Function that generates all the possible models with a set of variables
all_models <- function(variables){
  # How many variables in "variables"?
  K <- length(variables)
  # Use binary representation
  bin_vec <- rep(list(0:1), K)
  # Makes vectors of 1 and 0
  # Consider all of the different combinations, except the empty model.
  # There will be 2^K - 1 combinations
  bin_mat <- expand.grid(bin_vec)[-1, ]

  # Initialize the results. The loop will fill that list
  list_of_RHS <- list()
  # Fill up the list by looping over all combinations
  for(i in 1:nrow(bin_mat)){
    list_of_RHS[[i]] <- name_from_bin(bin_mat[i, ], variables)
  }
  return(list_of_RHS) # Each row of that list is a combination of covariates
}
```

Validation sets: Illustration in Rstudio

- Let us define all the variables we need for the RHS of the regressions

```r
# Show all the X's to consider. Here, we exclude column 1 and the "Balance" column as it is our Y
max_X <- colnames(Credit_1)[-c(1, which(colnames(Credit_1)=="Balance"))]
max_X
```

- Let us check some of the combinations (how many are there in total?)

```r
all_models(max_X)[12:15]
```

```
## [[1]]
## [1] "Rating" "Cards"
##
## [[2]]
## [1] "Income" "Rating" "Cards"
##
## [[3]]
## [1] "Limit"  "Rating" "Cards"
##
## [[4]]
## [1] "Income" "Limit"  "Rating" "Cards"
```

Validation sets: Illustration in Rstudio

- Finally, assemble all the functions into a single one, that will do all the steps above:

```r
# function that estimates all the possible models and computes the test MSE
all_subset_regression <- function(covariates_to_consider, y_var, train_dat, test_dat)
  {
  models_to_consider <- all_models(covariates_to_consider) # Makes all the possible combos

  # For each combo, run lm() and compute the training and test MSE
  # Map() is a function that loops over stuff in a more efficient way than "for"
  # It maps "models_to_consider" as "X_name" in the function "MSEs",
  # and we add the other arguments of the MSEs function
  results <- map(models_to_consider, MSEs, Y_name = y_var, training_data = train_dat,
                 test_data = test_dat)
  # The "results" will be a list of 3 columns.
  # First one is the number of X, second is the training MSE, third is the test MSE
  useful_results <- matrix(unlist(results), ncol = 3, byrow = TRUE) # Format the "results" nicely
  useful_results <- as_tibble(useful_results)
  names(useful_results) <- c(
    "num_vars",
    "training_error","test_error")
  return(useful_results)
}
```

Validation sets: Illustration in Rstudio

- Let's do it!

```r
performances <- all_subset_regression(covariates_to_consider = max_X,
                                      y_var = "Balance",
                                      train_dat = Credit_1,
                                      test_dat = Credit_2)
```

Validation sets: Illustration in Rstudio

- Now, let us check the training and test errors for each number of variables. Note that the smallest training error occurs for the maximum amount of variables

```
min_k_train <- performances %>%
  group_by(num_vars) %>% # Smallest training error per number of covariates used
  summarise(min_training_error = min(training_error))
min_k_train
```

```
## # A tibble: 10 x 2
##    num_vars min_training_error
##       <dbl>              <dbl>
##  1        1             57248.
##  2        2             30425.
##  3        3             10573.
##  4        4              9943.
##  5        5              9736.
##  6        6              9590.
##  7        7              9579.
##  8        8              9574.
##  9        9              9573.
## 10       10              9573.
```

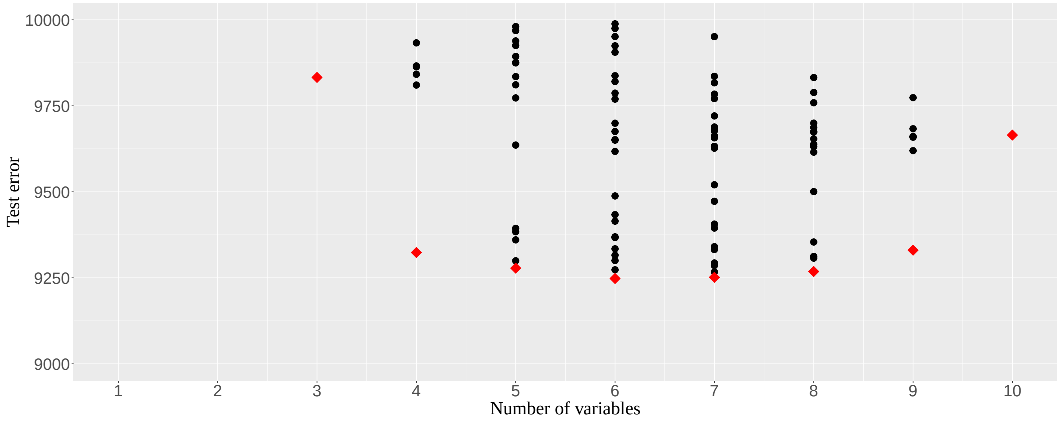
Validation sets: Illustration in Rstudio

- Now, let us check the training and test errors for each number of variables. Note that the smallest test error does not occur for the maximum amount of variables

```r
min_k_test <- performances %>%
  group_by(num_vars) %>% # Smallest test error per number of covariates used
  summarise(test_error = min(test_error))
min_k_test
```

```
## # A tibble: 10 x 2
##    num_vars test_error
##       <dbl>      <dbl>
##  1        1     50484.
##  2        2     24819.
##  3        3      9833.
##  4        4      9324.
##  5        5      9278.
##  6        6      9248.
##  7        7      9252.
##  8        8      9268.
##  9        9      9330.
## 10       10      9665.
```

Validation sets: Illustration in **Rstudio**



Validation sets: Illustration in Rstudio

- Which model has the lowest test error? Let us check

```
which(performances$test_error == min(performances$test_error))
```

```
## [1] 187
```

- That chunk of code does the same thing:

```
which.min(performances$test_error)
```

```
## [1] 187
```

Validation sets: Illustration in Rstudio

- Which model is it? What $\boldsymbol{X}$'s are in it?

```
performances[187, ]
```

```
## # A tibble: 1 x 3
##   num_vars training_error test_error
##      <dbl>          <dbl>      <dbl>
## 1        6          9792.      9248.
```

```
all_models(max_X)[[187]]
```

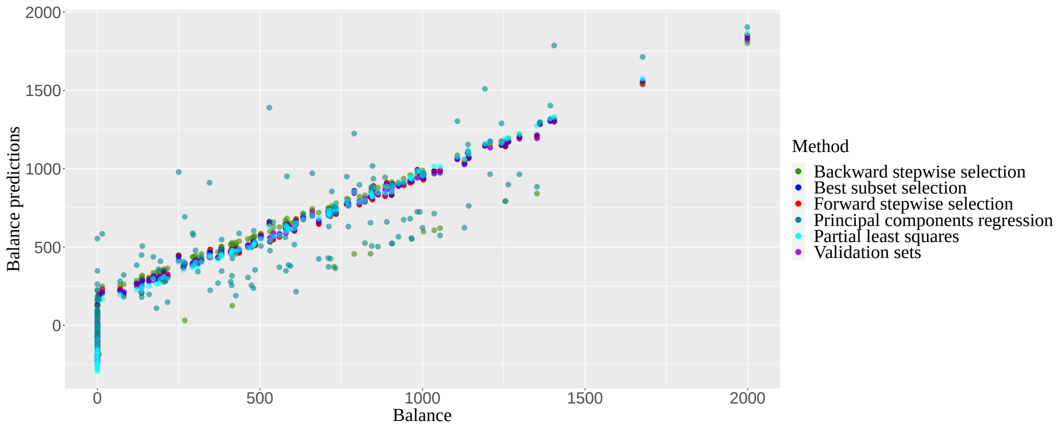
```
## [1] "Income"    "Limit"     "Cards"     "Age"       "Education" "Student"
```

Validation sets: Best model in Rstudio

```
##
## Call:
## lm(formula = gen_formula("Balance", best_combi), data = Credit_3)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -218.295  -72.541   -7.919   60.172  244.490
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -438.12186   51.17656  -8.561 3.15e-14 ***
## Income        -7.30618    0.40411 -18.080  < 2e-16 ***
## Limit          0.25443    0.00632  40.257  < 2e-16 ***
## Cards         25.85580    6.34983   4.072 8.15e-05 ***
## Age           -0.65909    0.52084  -1.265    0.208
## Education      0.85990    2.81284   0.306    0.760
## StudentYes   403.38847   31.73378  12.712  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 99.9 on 127 degrees of freedom
## Multiple R-squared:  0.9506, Adjusted R-squared:  0.9483
## F-statistic: 407.6 on 6 and 127 DF,  p-value: < 2.2e-16
```

Validation sets: Illustration in Rstudio

- Let us compare the different approaches on the 3^{rd} data set



Root Mean Squared Error (RMSE) with each approach

```r
# Subset selection procedures
best_subset_mse <- MSE(Credit_3$Balance, best_subset_pred)
forward_mse <- MSE(Credit_3$Balance, forward_stepwise_pred)
backward_mse <- MSE(Credit_3$Balance, backward_stepwise_pred)
# Dimension reduction methods
PCR_mse <-  MSE(Credit_3$Balance, pcr_pred)
PLSR_mse <- MSE(Credit_3$Balance, plsr_pred)
# Validation sets method
validation_mse <- MSE(Credit_3$Balance, Credit_3$validation_sets_pred)
```

Root Mean Squared Error (RMSE) with each approach

```
##                              Method      RMSE
## 1                       Best subset  95.99839
## 2                  Forward stepwise  96.47227
## 6                   Validation sets  97.25878
## 5             Partial least squares  99.56682
## 3                 Backward stepwise 147.00477
## 4 Principal components regression 257.05713
```

K-fold cross validation

- Similar to validation sets, the sample is cut into a **training** part and a **test** part
- Divide the data set in $\boldsymbol{K}$ groups, or **folds**
- Put the first fold aside, use the $\boldsymbol{K-1}$ other folds to estimate the models, test their predictive power on the first fold
- Repeat the same procedure with the second fold aside, and the first fold as part of the $\boldsymbol{K-1}$ folds
- For each model, we obtain $\boldsymbol{K}$ **test MSE**
- Choose the best model based on the **lowest average test MSE**
- The $\boldsymbol{K-1}$ folds are used as training data sets, the $\boldsymbol{K^{th}}$ fold as the test data set

K-fold cross validation

- How to choose $\boldsymbol{K}$? We could use cross validation…
- But that would introduce another parameter to optimize over. Talk about a rabbit hole
- It is common to use $\boldsymbol{K = 10}$
- If you have a model selection problem, and are not sure how to go about it, cross validation is **always** a good idea

Model averaging

- **Model selection** is important if one cares about estimating coefficients (many of the procedures shown above are used for linear models)
- If you only want to predict well, why would you care about choosing one variable over another?
- **Model averaging** takes the average of predictions over different models
- It shows better predictive performances that a single model
- Machine learning algorithms focus on prediction, so model averaging is used quite a lot
- Heard of the Netflix competition? Model averaging is what the winners used

Model averaging in action

- Let us average the predictions of the 4 best methods

```
##                            Method      RMSE
## 1                     Best subset  95.99839
## 2                Forward stepwise  96.47227
## 7          Model averaging (best 4)  96.52820
## 3                 Validation sets  97.25878
## 4           Partial least squares  99.56682
## 5               Backward stepwise 147.00477
## 6 Principal components regression 257.05713
```